Friedrich-Alexander-Universität
Faculty of Engineering

Dissertation

2024

Markus Holzer

Code generation in a lattice Boltzmann framework for exascale computing

Code generation in a lattice Boltzmann framework for exascale computing

Codegenerierung in einem Lattice-Boltzmann Programmpaket für Exascale Computing

DER TECHNISCHEN FAKULTÄT
DER FRIEDRICH-ALEXANDER-UNIVERSITÄT
ERLANGEN-NÜRNBERG
ZUR
ERLANGUNG DES DOKTORGRADES (DR.-ING.)

VORGELEGT VON

MARKUS HOLZER

AUS FREISING

Als Dissertation genehmigt
von der Technische Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: 29. November 2024

Gutachter: Prof. Dr. Ulrich Rüde
Prof. Dr. Christian Holm
Prof. Dr. Jonas Latt

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

délivré en co-tutelle avec Université d'Erlangen-Nuremberg

préparé à Toulouse INP

Generation de code automatique pour le calcul exaflopique pour la methode de boltzmann sur réseaux

---

Thèse présentée et soutenue, le 29 novembre 2024 par

Markus HOLZER

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
CERFACS

Thèse dirigée par
Catherine LAMBERT et Ulrich RÜEDE

Composition du jury
M. Jonas LATT, Rapporteur, Université de Genève
M. Christian HOLM, Rapporteur, University of Stuttgart
M. Eric CLIMENT, Examinateur, Toulouse INP
M. Tobias GÜNTHER, Examinateur, Friedrich-Alexander-Universität
M. Jens HARTING, Examinateur, Friedrich-Alexander-Universität
Mme Catherine LAMBERT, Directrice de thèse, Toulouse INP
M. Ulrich RÜEDE, Co-directeur de thèse, Friedrich-Alexander-Universität

Membres invités
M. Gabriel STAFFELBACH, ONERA

Abstract

Exascale supercomputers are computing systems capable of performing 10^{18} floating point operations per second. The supercomputer named Frontier first broke this barrier of one exaFLOPS and officially initiated the era of exascale computing in 2022. The immense scale of systems like this imposes significant challenges in developing codes that can fully exploit this computing power. Furthermore, the increasingly heterogeneous hardware employed by today's leading supercomputers adds another layer of complexity. In the field of computational fluid dynamics, it is therefore crucial to carefully consider every aspect of a numerical simulation, starting with the design and selection of algorithms suited to such environments. For example, algorithms like the lattice Boltzmann method are explicitly designed with massive parallelism in mind, making them a notable alternative to other more established methods. Nevertheless, a highly efficient implementation of this algorithm must be tailored to the respective hardware for optimal usage of resources.

To address these challenges, this thesis explores the use of code generation through an embedded domain-specific language. Code generation enables us to target specific hardware architectures and apply precise optimisations that leverage domain-specific knowledge. In this research, we extend and redesign the Python package LBMPY to support state-of-the-art variants of the lattice Boltzmann method. LBMPY represents the lattice Boltzmann method symbolically using a computer algebra system, allowing the automatic derivation of discretised equations based on user-defined specifications. To obtain equations with a minimal amount of floating-point operations we fundamentally enhance the simplification capabilities of LBMPY in this work. The discretised equations derived by LBMPY are provided to the Python package PYSTENCILS which generates highly optimised architecture-specific compute kernels in a lower-level language from these. We expand the range of supported hardware platforms and overhaul crucial aspects of the code generation process, such as the typing system, to improve performance and maintainability.

A sophisticated integration of these compute kernels into the massively parallel multiphysics framework WALBERLA is also developed, with an in-depth discussion of the key implementation components. One of the most significant advancements in this integration is the generation of highly specialised interpolation kernels. These kernels are essential for transferring information between cells of differing resolutions within the simulation domain, ensuring the accuracy and consistency of the data across varying grid sizes. This development has enabled us to perform the largest simulation run to date using the lattice Boltzmann method on a nonuniform domain, utilising more than 4000 AMD MI250X graphics processing units. The ability to efficiently manage such a vast and heterogeneous computational environment underscores the effectiveness of our approach in scaling complex simulations on next-generation hardware platforms.

We verify and validate our approach by simulating turbulent single-phase flow around a sphere using a nonuniform mesh configuration on graphics processing units, successfully reproducing the drag crisis—a complex phenomenon that occurs at Reynolds numbers above 200 000. Additionally, we demonstrate the capabilities of our method through slug flow simulations, offering new insights into the behaviour of Taylor bubbles in complex annular pipe configurations. Finally, we analyse the trajectories of droplets under the influence of a laser heat source in three-dimensional thermocapillary flows. To evaluate the performance of our approach, we present results from all these scenarios on the latest central processing unit and general purpose graphics processing unit hardware. We provide single-node performance

data and offer valuable insights by contextualising the measured results with appropriate performance models. Lastly, we examine the scalability of our developments by presenting both weak and strong scaling results on several of the world's leading supercomputers.

Résumé

Les superordinateurs Exascale sont des systèmes informatiques capables de réaliser 10^{18} opérations en virgule flottante par seconde, appellé un exaFLOPS. Le superordinateur Frontier a franchi pour la première fois la barrière d'un exaFLOPS et a officiellement ouvert l'ère du calcul exascale en 2022. L'immense échelle de systèmes tels que celui-ci impose des défis importants dans le développement de codes capables d'exploiter pleinement cette puissance de calcul. En outre, le matériel de plus en plus hétérogène utilisé par les principaux superordinateurs d'aujourd'hui ajoute une couche supplémentaire de complexité. Dans le domaine de la mécanique des fluides numérique, il est donc crucial d'examiner soigneusement chaque aspect d'une simulation numérique, en commençant par la conception et la sélection d'algorithmes adaptés à de tels environnements. Par exemple, des algorithmes tels que méthode de Boltzmann sont explicitement conçus avec un parallélisme massif à l'esprit, ce qui en fait une alternative notable à d'autres méthodes plus établies. Néanmoins, une mise en œuvre très efficace de cet algorithme doit être adaptée au matériel respectif pour une utilisation optimale des ressources.

Pour relever ces défis, cette thèse explore l'utilisation de la génération de code par le biais d'un langage intégré spécifique à un domaine. La génération de code nous permet de cibler des architectures matérielles spécifiques et d'appliquer des optimisations précises qui exploitent les connaissances spécifiques au domaine. Dans le cadre de cette recherche, nous étendons et remanions le paquet Python LBMPY pour prendre en charge les variantes de pointe de la méthode de Boltzmann. LBMPY représente la méthode de Boltzmann symboliquement en utilisant un système d'algèbre informatique, permettant la dérivation automatique d'équations discrétisées basées sur des spécifications définies par l'utilisateur. Pour obtenir des équations avec un minimum d'opérations en virgule flottante, nous améliorons fondamentalement les capacités de simplification de LBMPY dans ce travail. Les équations discrétisées dérivées par LBMPY sont fournies au paquet Python PYSTENCILS qui génère des noyaux de calcul spécifiques à l'architecture hautement optimisés dans un langage de niveau inférieur à partir de celles-ci. Nous élargissons la gamme des plates-formes matérielles prises en charge et révisons des aspects cruciaux du processus de génération de code, tels que le système de typage, afin d'améliorer les performances et la maintenabilité.

Une intégration sophistiquée de ces noyaux de calcul dans le cadre multiphysique massivement parallèle WALBERLA est également développée, avec une discussion approfondie des composants clés de la mise en œuvre. L'une des avancées les plus significatives de cette intégration est la génération de noyaux d'interpolation hautement spécialisés. Ces noyaux sont essentiels pour le transfert d'informations entre des cellules de résolutions différentes au sein du domaine de simulation, garantissant la précision et la cohérence des données sur des grilles de tailles différentes. Ce développement nous a permis d'effectuer la plus grande simulation à ce jour en utilisant la méthode de Boltzmann sur un domaine non uniforme, en utilisant plus de 4000 AMD MI250X processeur graphique. La capacité à gérer efficacement un environnement de calcul aussi vaste et hétérogène souligne l'efficacité de notre approche dans la mise à l'échelle de simulations complexes sur des plates-formes matérielles de nouvelle génération.

Nous vérifions et validons notre approche en simulant un écoulement monophasique turbulent autour d'une sphère à l'aide d'une configuration de maillage non uniforme sur processeurs graphiques, reproduisant avec succès la crise de traînée - un phénomène complexe qui se produit à des nombres de Reynolds supérieurs à 200 000. En outre, nous démontrons les capacités de notre méthode par le biais de simulations d'écoulements de boue, offrant de nouvelles perspectives sur le comportement des bulles de Taylor dans des configurations de tuyaux annulaires complexes.

Enfin, nous analysons les trajectoires des gouttelettes sous l'influence d'une source de chaleur laser dans des écoulements thermocapillaires tridimensionnels. Pour évaluer les performances de notre approche, nous présentons les résultats de tous ces scénarios sur les processeurs et processeurs graphiques les plus récent. Nous fournissons des données de performance pour un seul nœud et offrons des informations précieuses en contextualisant les résultats mesurés avec des modèles de performance appropriés. Enfin, nous examinons l'évolutivité de nos développements en présentant des résultats d'évolutivité faibles et forts sur plusieurs des principaux superordinateurs du monde.

Zusammenfassung

Exascale-Supercomputer sind Computer, die mehr als 10^{18} floating point operations per second ausführen können. Der Supercomputer mit dem Namen Frontier durchbrach erstmals diese Grenze von einem ExaFLOPS und leitete 2022 offiziell die Ära der Exascale-Rechner ein. Die immense Größe solcher Systeme stellt die Entwicklung von Computerprogrammen, die diese Rechenleistung voll ausschöpfen können, vor große Herausforderungen. Darüber hinaus sorgt die zunehmend heterogene Hardware, die in den führenden Supercomputern von heute zum Einsatz kommt, für eine zusätzliche Komplexitätsebene. Im Bereich der numerischen Strömungsmechanik ist es daher von entscheidender Bedeutung, jeden Aspekt einer numerischen Simulation sorgfältig zu berücksichtigen, angefangen bei der Entwicklung und Auswahl von Algorithmen, die für solche Umgebungen geeignet sind. So sind beispielsweise Algorithmen wie die Lattice-Boltzmann Methode (LBM) ausdrücklich auf massive Parallelität ausgelegt, was sie zu einer attraktiven Alternative zu anderen, etablierteren Methoden macht. Dennoch muss eine hocheffiziente Implementierung dieses Algorithmus auf die jeweilige Hardware zugeschnitten sein, um die Ressourcen optimal zu nutzen.

Um diese Herausforderungen zu bewältigen, wird in dieser Arbeit der Einsatz von Codegenerierung durch eine eingebettete domänenspezifische Sprache untersucht. Die Code-Generierung ermöglicht es uns, spezifische Hardware-Architekturen anzuvisieren und präzise Optimierungen anzuwenden, die domänenspezifisches Wissen über diese nutzen. In dieser Arbeit wird das Python-Paket LBMPY erweitert und neu konzipiert, um moderne Varianten der LBM zu unterstützen. LBMPY stellt die LBM symbolisch mit Hilfe eines Computeralgebrasystems dar und ermöglicht die automatische Ableitung von diskretisierten Gleichungen auf der Grundlage von benutzerdefinierten Spezifikationen. Um Gleichungen mit einer minimalen Anzahl von Fließkommaoperationen zu erhalten, haben wir in dieser Arbeit die Vereinfachungsmöglichkeiten von LBMPY grundlegend verbessert. Die von LBMPY abgeleiteten diskretisierten Gleichungen werden dem Python-Paket PYSTENCILS zur Verfügung gestellt, das daraus hochoptimierte, architekturspezifische Programmteile in einer hardwarenahen Sprache erzeugt. Wir erweitern die unterstützten Hardwareplattformen und überarbeiten entscheidende Aspekte des Codegenerierungsprozesses, wie z.B. das Typisierungssystem, um die Leistung und Wartbarkeit zu verbessern.

Darüber hinaus wird eine ausgereift Integration dieser Programmteile in das massiv-parallele Multiphysik-Programmpaket WALBERLA entwickelt, wobei die wichtigsten Implementierungskomponenten eingehend erörtert werden. Einer der wichtigsten Fortschritte bei dieser Integration ist die Generierung von hochspezialisierten Programmteilen zur Interpolation. Diese Programmteile sind für die Übertragung von Informationen zwischen Zellen mit unterschiedlichen Auflösungen innerhalb des Simulationsbereichs zuständig und gewährleisten die Genauigkeit und Konsistenz der Daten über verschiedene Auflösungen hinweg. Diese Entwicklung hat es uns ermöglicht, den bisher größten Simulationslauf mit der LBM auf einem verfeinerten Gebiet durchzuführen, bei dem mehr als 4000 AMD MI250X Grafikprozessoren gleichzeitig genutzt wurden. Die Fähigkeit, einen so großen und heterogenen Rechner effizient zu verwalten, unterstreicht die Effektivität unseres Ansatzes bei der Skalierung komplexer Simulationen auf Hardware der nächsten Generation.

Wir verifizieren und validieren unseren Ansatz durch die Simulation einer turbulenten einphasigen Strömung um eine Kugel unter Verwendung eines verfeinerten Gitters auf Grafikprozessoren, wobei wir erfolgreich das Eiffel paradox reproduzieren - ein komplexes Phänomen, das bei Reynoldszahlen über 200 000 auftritt. Darüber hinaus demonstrieren wir die Fähigkeiten unserer Methode anhand von Strömungssimulationen, die neue Einblicke in das Verhalten von Taylor-Blasen in komplexen ringförmigen Rohrkonfigurationen bieten. Schließlich analysieren wir die Bahn von

Tröpfchen unter dem Einfluss einer Laserwärmequelle in dreidimensionalen Thermokapillarströmungen. Um die Leistung unseres Ansatzes zu bewerten, präsentieren wir Ergebnisse aus all diesen Szenarien auf der neuesten Prozessoren und Grafikprozessoren. Wir liefern Benchmarks für einzelne Programmteile und bieten wertvolle Einblicke, indem wir die gemessenen Ergebnisse mit geeigneten Modellen in Kontext bringen. Schließlich untersuchen wir die Skalierbarkeit unserer Entwicklungen, indem wir sowohl schwache als auch starke Skalierungsergebnisse auf mehreren der weltweit größten Supercomputern präsentieren.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Ulrich Rüde, who gave me the chance to work on this exciting topic and motivated me in the best way I could imagine. I am more than thankful that I could seek guidance whenever I needed it, while on the other side, I always had enough freedom to follow the paths that seemed exciting to me. Through his suggestion, I went to France to follow this research at CERFACS, which was the best decision of my life. I also thank Dr. Catherine Lambert, who gave me the opportunity to work at CERFACS.

The next person I would like to express my deepest gratitude to is Dr. Gabriel Staffelbach, who supervised me at CERFACS. I am thankful for every stimulating discussion about the intricacies of high-performance computing. It was a pleasure to work under your guidance, which extended my knowledge in every aspect. I also thank Prof. Ulrich Rüde, Dr. Gabriel Staffelbach, Prof. Jonas Latt and Prof. Christian Holm for examining my thesis.

I am obliged to Martin Bauer, who introduced me to the topic of code generation in my master's thesis. The ongoing discussions during my thesis helped me expand my ideas and develop the framework in every possible way. Furthermore, without his initial efforts, this work would not even exist.

During my thesis, I had the opportunity to work closely with the team at the Chair for System Simulation. I am immensely thankful for this. The whole team is just excellent and produces a genuinely fruitful atmosphere. From the waLBerla team, I would like to thank especially Philipp Suffa, who tackled the SCALABLE project with me, Helen Schottenhamml, who helped me to better understand the physics behind the simulations, and Jan Hönig, who developed the new type system of PyStencils with me. Around the PyStencils team, I am incredibly thankful to Michael Kuron. Without his insights, this thesis would not be half as good as it turned out now.

Furthermore, I would like to acknowledge the support of my students, Daniel Bauer and Frederik Hennig. Frederik Hennig worked with me for many years, and numerous ideas for this thesis have been developed with him.

On the CERFACS side, I would like to express my deepest gratitude to the whole ALGO-COOP team. I am more than thankful that I could have been a part of this team. Every day at CERFACS, I feel welcome, and I can seek out help if I need it.

A PhD procedure between two countries causes a significant bureaucratic overhead. Without the help of Alexandra Lukas–Rother, Iris Weiß at the FAU, Michele Campassens, and Brigitte Yzel at CERFACS, this would not have been possible. To all, I would like to express my deepest gratitude.

I am thankful to all the scientists with whom I had the delight of collaborating during this thesis. My deepest gratitude goes to Travis Mitchell, whose collaboration led to many ideas and successful publications.

A special thanks goes to all my proofreaders, who significantly increased the quality of this work. These are Helen Schottenhamml, Dr. Christoph Schwarzmeier and Dr. Michael Kuron. Thank you so so much; without your help, this work would not be the same.

Finally, I would like to express my highest gratitude to my family and friends, my mother, Siglinde Holzer, who always supported me in every step of my life, and my brother Matthias Holzer, who is the best brother I could ever wish for. Last but surely not least, I thank my wife, Pamela Holzer, from the deepest of my heart. This thesis was the most challenging step that I have taken in my life so far. Without the unconditional love, the encouragement and the unlimited support that you gave me at every moment of my work, I would not have been able to have the stamina until the end.

My gratitude also goes to the SCALABLE (https://scalable-hpc.eu/) project, which was the primary financial resource of this thesis. The SCALABLE project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956000. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Germany, and the Czech Republic. I also gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC). I acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Regular Access call. I gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High-Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

Contents

List of Figures

List of Acronyms

AABB
axis-aligned bounding box 58

ACE
Allen-Cahn equation 147

ADE
advection-diffusion equation 42

AMR
adaptive mesh refinement 60

AoS
array of structures 44

API
application programming interface 5

AST
abstract syntax tree 42

AVX512
512-bit Advanced Vector Extensions 128

BGK
Bhatnagar-Gross-Krook 15

BTE
Boltzmann transport equation 10

CACE
conservative Allen-Cahn equation 26

CACM
conservative Allen-Cahn model 4

CFD
computational fluid dynamics vii

CPU

CHAPTER 1

INTRODUCTION

Contents

From a general motivation in Section 1.1 where we discuss the fundamental driving points of this work, we derive the main research goals and put them in the perspective of previous work in Section 1.2. Afterwards, we address the main research contributions and the outline of the remaining thesis in Section 1.3 and Section 1.4. Finally, in Section 1.5, we provide important guidance on how other researchers can reproduce the results presented in this work.

1.1 Motivation

Computational systems able to perform at least 10^{18} floating point operations per second (FLOPS) (one exaFLOPS) are called exascale systems. Such a system can perform one floating-point operation every billionth of a billionth of a second, also known as Attosecond (10^{-18} s). Coincidentally, this is the same timescale on which the rapid processes of electrons that underlie all computing hardware can be observed. Recently, groundbreaking research in this field was awarded the Nobel Prize in Physics. As Eva Olsson, chair of the Nobel Committee for Physics, expressed it: "We can now open the door to the world of electrons [12]." Similarly, electronics engineers open the doors to smaller and smaller transistors. For example, IBM plans to produce a chip containing transistors of the size of two nanometres (nm) by 2025. This is about five atoms in size, and their goal is to put 50 billion ($50 \cdot 10^9$) transistors on a single chip [13].

This indeed is a tremendous achievement, however, it should not hide the fact that the transistor size is slowly reaching its theoretical limitations. Likewise, the bandwidth between the processor and the main memory is not growing at the same rate as the chips advance. This phenomenon is often called the *memory wall* and poses one of the toughest challenges for engineers in the exascale computing era [14, 15]. It is, therefore, not surprising that these issues are urgently discussed in a task force report recently published by the Society for Industrial and Applied Mathematics (SIAM) [16]. They conclude in their report [16]:

> *High performance computing (HPC) is a key element of computational science, but HPC itself is at an inflection point. Historical drivers for performance improvement have run their course, with transistors now nearly as small as they can get and supercomputer energy consumption at a practical maximum, making the energy-efficiency of future HPC platforms of paramount concern. The only viable way to continue improving performance will involve architectural specialization and heterogeneous supercomputers.*

While significant research still needs to be done to fully utilise the potential of more exotic architectures like quantum accelerators, it is already clear that heterogeneous supercomputers dominate the market. Most prominently, at the time of writing, nine of the top ten supercomputers in the Top 500 list[1] use accelerators in the form of general purpose graphics processing units (GPGPUs) to achieve their massive performance. Unfortunately, these accelerators are produced by different hardware vendors, and no unified interface exists to program them efficiently. Instead, as pointed out by Andreas Herten [17], each vendor features native and derived programming models and not all models are supported on all platforms. He identified nine different approaches to programming GPGPUs from the three hardware vendors in his work. Remarkably, at this

[1] https://top500.org/lists/top500/list/2024/06/

point, the standard library[2] of C++ supports offloading to some GPGPUs natively. However, the standard library does not fully support for GPGPUs produced by AMD, which are the accelerators used in the first exascale system, Frontier[3], and the number one supercomputer in Europe named Lumi[4]. This example showcases that the landscape for computer architectures and their respective programming interfaces is diverse and is especially likely to become increasingly diverse. This fact is one of the main driving points for this thesis, as the mentioned situation might lead to overly complex, repetitive and, finally, hard-to-maintain software. This can be the case, especially when software is ported by hand using the respective lower-level language to address the hardware. Hence, software design for diverse heterogeneous architectures will be recurring during this work.

Among many other use cases, typical applications for large-scale supercomputers come from the field of computational fluid dynamics (CFD) as complex industrial applications and natural flow phenomena usually require an extremely fine resolution to predict the behaviour of the flow field correctly. In cases where experiments are either too expensive, too dangerous or even impossible to perform, data from simulations is needed. Classically, flow phenomena are predicted by solving the Navier-Stokes equations (NSEs) numerically. However, numerical schemes to solve the NSEs for complex, real-world applications can be computationally prohibitive. One reason for this is that the NSEs contain a nonlinear advection term, which typically leads to complex iterative schemes that need to be resolved in every timestep [18].

Over the past three decades, the lattice Boltzmann method (LBM) has emerged as an attractive alternative to traditional NSE solvers, especially for unsteady and multiphase flow problems [19–21]. In the LBM, nonlinear expressions are confined in its collision operator, which does not depend on information from neighbouring cells. On the other hand, dependencies with neighbouring cells are solved in a streaming step, a simple linear operator that does not rely on complex computations. Explicit time-stepping schemes are oftentimes easier to parallelise on a large scale, as they do not involve the solution of a linear system. The most common variants of the LBM use an explicit time-stepping scheme, making them well-suited for massively parallel architectures. Nevertheless, numerous variants of the LBM exist with distinct advantages and disadvantages. For example, the mentioned collision operator can be solved in various more or less complex fashions with different accuracy and efficiency.

Consequently, a second main problem arises that we aim to address in this thesis. In contrast to the portability problem given by the heterogeneous architecture landscape, here the problem arises entirely from the algorithmic side. Ideally, different variants of the LBM are formulated in a unified framework with minimal repetition. Previously, in the multiphysics framework WALBERLA, this problem was addressed by heavily relying on static polymorphism. This means that distinct parts of the LBM are separated into building blocks that form a complete compute kernel at the compilation stage based on a configuration by the user. However, static polymorphism strongly increases the framework's complexity; thus, only a few distinct building blocks exist, limiting the range of possibilities. For example, the streaming step is fixed to one variant and is not changeable if one wants to.

[2] https://en.cppreference.com/w/cpp/header/algorithm
[3] https://www.olcf.ornl.gov/frontier/
[4] https://www.lumi-supercomputer.eu/

This problem amplifies severely in conjunction with the first problem. In the past, critical parts of these building blocks[5] have been specialised to target specific central processing unit (CPU) architectures. In this way, maximum reachable performance and excellent scalability could be shown in several works [22–24]. However, supporting more complex state-of-the-art variants of the LBM and accommodating a broader range of architectures, such as GPGPUs, would significantly increase the complexity of WALBERLA.

To target both problems, it is crucial to develop methods to simplify the implementation and maintenance of highly optimised LBM codes without sacrificing efficiency. This thesis explores innovative strategies and tools that can bridge the gap between ease of use and high performance. An emerging technology that has the potential to mitigate the outlined problems is using a domain-specific language (DSL) in conjunction with code generation techniques. A DSL has the advantage that the problem can be described in an abstract way perfectly tailored to the problem's context. Then, architecture-specific implementations can be generated from this description in an automated fashion. In this way, scientists do not need to know details about the intrinsic complexity of specialised optimisations, and these can be applied to all supported methods. Since DSLs are tailored to specific problems, optimisations that combine the domain and architecture knowledge can be introduced.

1.2 Background and Objectives

Initially, the WALBERLA multiphysics framework was designed as an HPC framework in C++[6]. Due to the complexities of supporting various architectures and specialised algorithms, code generation techniques have been introduced to WALBERLA recently [25]. While this approach showed promising results, it was only used on benchmark cases so far. This thesis aims to extend the code generation approach used in the WALBERLA framework for using the latest supercomputers in complex single and multiphase flow applications.

To generate efficient compute kernels for the LBM, a modular approach is employed in combination with the WALBERLA framework. In this setup, the Python package LBMPY expresses the LBM symbolically and derives discretised equations from high-level descriptions. These discretised equations are then transformed to architecture-specific compute kernels in a lower-level programming language by the Python package PYSTENCILS. Finally, WALBERLA can integrate these compute kernels to extend the existing framework.

At the start of this thesis, LBMPY supported standard moment-based collision operators, the most important boundary conditions and force transformation schemes. A detailed overview of its original functionality was published by Bauer *et al.* [26]. Furthermore, to simulate immiscible fluids with high-density contrasts under high Reynolds numbers, a first implementation of the conservative Allen-Cahn model (CACM) was realised in the author's master's thesis [27].

[5] mostly the collision model
[6] https://walberla.net/

On the other hand, PySTENCILS supported optimisations like spatial blocking, loop reordering and loop splitting. For the efficient usage of CPU architectures, support for x86 single instruction, multiple data (SIMD) instruction sets was provided, and shared memory usage was supported by OpenMP pragmas in the generated compute kernels. The support for NVIDIA GPGPUs was realised by the compute unified device architecture (CUDA) backend of PySTENCILS. Bauer *et al.* provided more details on the original functionality in [28].

Furthermore, Bauer *et al.* equipped waLBERLA with a basic integration of PySTENCILS and LBMpy to realise large-scale simulations [25, 26]. The core concept of this integration is to provide template files within the waLBERLA framework into which generated compute kernels are inserted. In this fashion, it is possible to use the resulting files directly in the waLBERLA framework due to the consistent application programming interface (API) used in the generated files. With this workflow, LBM simulations on uniform grids could be executed on CPU and NVIDIA GPGPU architectures. The largest demonstrated benchmark cases consisted of $4.4 \cdot 10^{11}$ cells on the SuperMUC-NG [7] cluster and $3.4 \cdot 10^{10}$ cells on the PizDaint [8] supercomputer [25, 26].

1.3 Main Contributions

From the motivation, the background and the main driving points of the thesis, the following main research contributions have been realised:

- We significantly improved and extended the LBMpy package to support state-of-the-art collision operators needed for highly turbulent flow regimes and multiphase flows. We also implemented advanced boundary conditions to support outflow boundaries without reflections and introduced interpolated no-slip boundary conditions for increased accuracy on complex geometries.
- We extended the PySTENCILS package to a wider range of architectures including AMD GPGPUs. In the same effort, we strengthened the typing system to allow the generation of more complex compute kernels.
- We extended the integration of the code generation pipeline with waLBERLA to support grid transitions with different resolutions. We improved the static mesh refinement algorithm in waLBERLA and ported it to GPGPUs architectures.
- To demonstrate the capabilities of these new developments, we successfully captured the drag crisis of a spherical object, showcasing our ability to simulate highly turbulent flow problems with complex geometries using a nonuniform mesh configuration.
- We validated and extended challenging industrial multiphase flow problems within the slug flow regime. Our experiments reveal the influence of annulus pipe geometries on the shape of Taylor bubbles.
- We validated and extended thermocapillary flow problems. Based on findings in the literature on two-dimensional configurations, we show that full three-dimensional simulations expose a much richer behaviour.

[7] https://doku.lrz.de/supermuc-ng-10745965.html
[8] https://www.cscs.ch/computers/piz-daint

- We carefully benchmarked all developments on single-node configurations to demonstrate excellent base performance and conducted large-scale benchmark cases on the latest pre-exascale supercomputers.

- We implemented all developements in the open-source frameworks LBMPY, PYSTENCILS and WALBERLA. This opens the door for future collaborations and accelerates innovations in the community.

1.4 Outline and Thesis Structure

The structure of the thesis is tightly bound to the research objectives. In Chapter 2, we introduce the numerical foundations of the thesis. This means a brief introduction to the lattice Boltzmann method and the most important algorithmic building blocks used in this thesis. From there, we give the necessary background and details in Chapter 3 to extend the LBM for the simulation multiphase flows. Since there are many different models to realise multiphase flows, we also briefly compare the most common models in the literature. The software stack that builds the core part of this thesis is introduced in Chapter 4. In this chapter, we introduce the PYSTENCILS and LBMPY packages and show their integration in the WALBERLA framework. Contributions realised as part of this thesis are carefully explained and discussed in the context of previous developments. Afterwards, Chapter 5 details the mesh refinement algorithm. We give a detailed comparison with the previous developments from Florian Schornbaum [29] to highlight significant improvements realised in this work. To showcase the applicability of our new advancements, we have analysed three distinct applications that vastly profit from the developments introduced before. In the first application, we simulate highly turbulent flow to capture the drag crisis of a sphere in a flow channel in Chapter 6. Due to the high resolution needed for this problem, it is an ideal case to show the capabilities of the newly developed mesh refinement approach on GPGPUs. This follows the simulation of slug flow problems in Chapter 7, where our new developments capture the behaviour of Taylor bubbles in previously published annulus pipe geometries. Afterwards, new regimes are explored to build on the existing research. An extension of the phase-field multiphase model to thermocapillary flows is presented in Chapter 8. The main contribution of this chapter is to capture the behaviour of droplets in a Couette channel flow under the influence of laser heat sources. Lastly, the performance of our code is benchmarked in Chapter 9 using carefully crafted cases. The performance results are shown on single-node configurations and large-scale weak- and strong-scaling scenarios. The thesis finished with a conclusion presented in Chapter 10.

1.5 Reproducability

In scientific research, it is essential that others can reproduce the results of previous studies. However, reproducing simulation runs requires careful attention to many important details. This process begins with ensuring the correct version of the source code and continues with the build chain used to compile the code. In this work, the build chain includes a Python environment for the code generator, which may also influence the generated code. During code execution, the system configuration plays a pivotal role, particularly in performance benchmarks.

To ensure reproducibility, several measures have been implemented. The first key measure is that all source code used to produce the results in this thesis is open source. It is important to note that the results were produced over several years of development. During this time, all packages have undergone multiple software versions. However, LBMPY, PYSTENCILS, and WALBERLA employ continuous integration within *GitLab* to ensure that previous developments are preserved correctly. For all applications developed in this thesis, unit tests, integration tests, and compilation tests have been provided to the continuous integration system to ensure that future developments do not affect the results. Additionally, within LBMPY and PYSTENCILS, the integration with WALBERLA is tested, meaning that the compute kernels for all applications in this work are generated and compiled when the code is updated on the respective repository.

The versions of LBMPY and PYSTENCILS used in this work are 1.3.6 for both packages. These versions are available on *PyPi*[9,10]. For WALBERLA, version 7.0 is the latest version that contains all the simulation codes, and it can be downloaded from the *GitLab* repository[11].

A vital tool to increase reproducibility is *MachineState*, developed by the HPC group at the University of Erlangen-Nürnberg [30]. This tool allows users to take a snapshot of the system configuration, including compiler versions, CPU frequencies, and critical information about the operating system. These snapshots can be compared with those from later runs to ensure that the system configuration remains consistent. Any changes are highlighted for the user. In this work, version 0.5.0 of *MachineState*[12] was used for the benchmark runs.

[9] https://pypi.org/project/pystencils/
[10] https://pypi.org/project/lbmpy/
[11] https://i10git.cs.fau.de/walberla/walberla
[12] https://pypi.org/project/MachineState/

CHAPTER 2

FUNDAMENTALS OF THE LATTICE BOLTZMANN METHOD

Contents

This chapter provides the numerical foundation of lattice Boltzmann method. A short overview of its origins in the Boltzmann transport equation is shown in Section 2.1, followed by a modern description of the lattice Boltzmann method using generating functions in Section 2.2. This mathematical framework, forming the heart of the code generator LBMPY*, was previously published [1] in the scope of this thesis. Within this mathematical framework, state-of-the-art moment-based collision models, along with various streaming patterns, are introduced. Afterwards, a short overview of boundary conditions and force models relevant to the scope of this work is given. The chapter closes by explaining the full algorithm to simulate single-phase flow problems with the lattice Boltzmann method in Section 2.3*

2.1 The Boltzmann Equation

The foundation of the lattice Boltzmann method (LBM) goes back to the kinetic theory of gases (KTG), which was developed mainly by the contributions of Maxwell and Boltzmann [31, 32]. Their basic idea was that fluids, although consisting of individual particles, follow well-defined rules. An explanation for this behaviour needs to find its roots in probability theory and is summarised in the Boltzmann transport equation (BTE)

$$\frac{\partial f}{\partial t} + \sum_{\alpha} \xi_{\alpha} \frac{\partial f}{\partial x_{\alpha}} + \sum_{\alpha} \frac{F_{\alpha}}{\rho_f} \frac{\partial f}{\partial \xi_{\alpha}} = \Omega\left(f\right). \tag{2.1}$$

The BTE describes the transport of groups of particles $f \equiv f\left(\boldsymbol{x}, \boldsymbol{\xi}, t\right)$ called particle distribution functions (PDFs). It represents the density of particles with a given velocity $\boldsymbol{\xi}$ at position $\boldsymbol{x}$ and time t. The gradient operators in the BTE correspond to the geometrical and the velocity space in which the PDF is advected. The right-hand side defines the redistribution of the particles due to local collision between particles. Thus, the collision operator is $\Omega\left(f\right)$. To link the motion of particles with the macroscopic description of fluids, i.e., describing the fluid by macroscopic quantities like density or velocity, a statistical observer is necessary [33]. The most common statistical observers are the moments of the PDFs. The moments are integral quantities of the PDFs weighted with a function of the mesoscopic velocity $\boldsymbol{\xi}$. For example, the zeroth order moment reveals the fluid's mass density

$$\rho\left(\boldsymbol{x}, t\right) = \int f\left(\boldsymbol{x}, \boldsymbol{\xi}, t\right) d^3\xi. \tag{2.2}$$

Similarly, the fluid's momentum density can be found in the first-order moments

$$\rho\left(\boldsymbol{x}, t\right) \boldsymbol{u}\left(\boldsymbol{x}, t\right) = \int \boldsymbol{\xi} f\left(\boldsymbol{x}, \boldsymbol{\xi}, t\right) d^3\xi, \tag{2.3}$$

which can be understood as considering the particles' contribution $\boldsymbol{\xi} f$ to the momentum density.

Solving the BTE directly proposes severe difficulties due to the complexity of the collision operator. This goes back to the work of Boltzmann himself [32]. The original idea was to simplify the collision operator by binary particle collisions. However, considering all possible outcomes of particle pairs leads to a complex double integral over the velocity space, thus not providing the desired simplicity to solve the equation. For this reason, the first simplified collision operator was proposed by the work of Bhatnagar, Gross, and Krook [34]. Their work makes the assumption of dense fluids by introducing a relaxation

time τ to simplify the collision process drastically:

$$\boldsymbol{\Omega}^{\mathrm{BGK}} = -\frac{1}{\tau}\left(\boldsymbol{f} - \boldsymbol{f}^{\mathrm{eq}}\right). \tag{2.4}$$

With this collision model, all particles move towards an equilibrium state that is described by the Maxwell-Boltzmann equilibrium

$$f_{\mathrm{MB}}^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \boldsymbol{\xi}\right) = \frac{\rho}{(2\pi RT)^{\frac{D}{2}}} \exp\left[-\frac{(\boldsymbol{\xi} - \boldsymbol{u})^2}{2RT}\right], \tag{2.5}$$

where D is the number of spatial dimensions, R is the gas constant, and T is the temperature.

2.2 The Lattice Boltzmann Equation

To obtain a numerical method for solving the BTE, a discretisation of the velocity space is applied [19, 35]. Thus the velocity $\boldsymbol{\xi}$ of the PDF can only take discrete values $\boldsymbol{c}_i = \left(c_{ix}, c_{iy}, c_{iz}\right)$. Furthermore, the space is discretised on a Cartesian computational grid called lattice with a grid spacing of Δx, and the time t is discretised using a time step size Δt. This leads to the lattice Boltzmann equation (LBE)

$$\boldsymbol{f}\left(\boldsymbol{x} + \boldsymbol{c}_i, t + \Delta t\right) = \boldsymbol{\Omega}\left(\boldsymbol{f}\left(\boldsymbol{x}, t\right)\right) + \boldsymbol{f}^F. \tag{2.6}$$

Here, $\boldsymbol{f}^F$ represents a source term that is further explained in Section 2.2.4. In computational fluid dynamics (CFD), the LBE plays a special role as it forms a promising alternative to classical discretisation approaches for the Navier-Stokes equations (NSEs) based, e.g., on a finite-differences or finite-volume formulation [33, 36]. The NSEs for incompressible Newtonian fluids can be stated as,

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.7}$$

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \left(\boldsymbol{u} \cdot \nabla\right)\boldsymbol{u}\right) = -\nabla p + \mu \nabla^2 \boldsymbol{u} + \boldsymbol{F}_b, \tag{2.8}$$

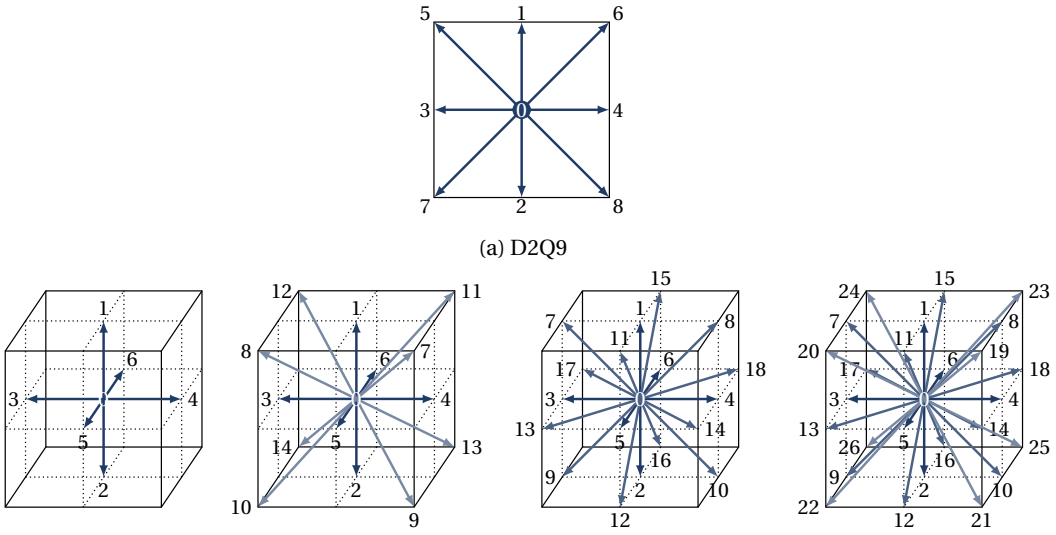
where $\boldsymbol{F}_b$ includes external forces, e.g., gravity as shown in Equation (2.34). However, the LBE is not limited to simulating physical phenomena as expressed by the NSEs. In fact, the LBE can be used to simulate first-order partial differential equations (PDEs) derived from the linearised BTE [21]. Commonly, the velocity discretisation of the PDF is given in DdQq notation where q defines the number of discrete velocities c and d represents the number of spatial dimensions. The lattice stencils used in this work are the D3Q7, D3Q15, D3Q19, and the D3Q27 stencil pictured in Figure 2.1.

Note that Equation (2.6) can be split into two parts

$$\boldsymbol{f}^* = \boldsymbol{\Omega}\left(\boldsymbol{f}\left(\boldsymbol{x}, t\right)\right) + \boldsymbol{f}^F \tag{2.9}$$

$$\boldsymbol{f}\left(\boldsymbol{x} + \boldsymbol{c}_i, t + \Delta t\right) = \boldsymbol{f}^*, \tag{2.10}$$

where $\boldsymbol{f}^*$ marks the post-collision PDFs, and the left-hand side represents the linear streaming step in which the discrete particles travel to neighbouring cells according to the lattice stencil. The streaming step can be understood as the advection term. On the other side, the right-hand side is the non-linear collision operator, which only needs cell



(a) D2Q9

(b) D3Q7 (c) D3Q15 (d) D3Q19 (e) D3Q27

Figure 2.1: Illustration of different lattice stencils for the LBM.

local information. Thus, the complex non-linear computations are local, and the more straightforward linear streaming step is non-local. This key feature sets the foundation of one of the greatest advantages of the LBM contrary to classical discretisations of the NSEs where the advection is non-linear and non-local [21, 33]. Furthermore, it is important to note that a minimal set of discrete velocities is required to recover specific physical phenomena. For example, only the D3Q19 and the D3Q27 stencils are suited to recover the NSEs while stencils like the D3Q7 and the D3Q15 are still enough to simulate advection-diffusion systems as shown in Chapter 3 [33].

Owing to the discretisation of the velocity space, computing the macroscopic quantities results in discrete sums contrary to the integrals presented in Equations (2.2) and (2.3). Hence, the sum over a cell's population set forms the density, while the weighted sum with respect to the discrete velocities forms the macroscopic velocity:

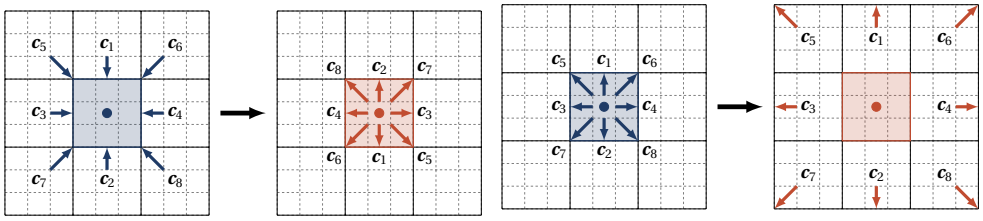
$$\rho(\boldsymbol{x},t) := \sum_{i=0}^{q-1} f_i(\boldsymbol{x},t), \qquad \boldsymbol{u}(\boldsymbol{x},t) := \frac{1}{\rho}\sum_{i=0}^{q-1} f_i(\boldsymbol{x},t)\,\boldsymbol{c}_i. \tag{2.11}$$

Here, it is important to note that the quantities obtained from the moments vary depending on the formulation of the LBE (see Chapter 3).

2.2.1 Streaming Patterns

Streaming the PDFs from one cell to another can be done in various ways. The most common method is to use a so-called two-grid algorithm [37]. Common examples of this kind of streaming pattern are the pull pattern shown in Figure 2.2a and the push pattern shown in Figure 2.2b. For simplicity reasons, the streaming patterns discussed in this section are shown in two dimensions. However, the extension to the three-dimensional velocity set is trivial. The pull-pattern follows the idea that PDFs are pulled from the neighbouring cells of a source grid. Then, in the centre cell, the collision operation is performed, and the newly obtained PDFs are stored on a destination grid. In the push-pattern, the collision operation is performed on the PDFs of the centre cell of a source grid.

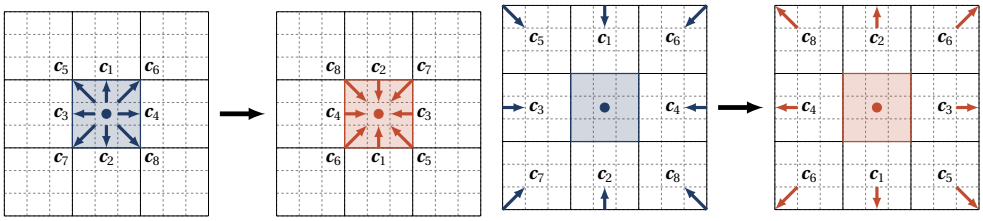
Afterwards, the updated PDFs get pushed to the neighbouring cells on a destination grid. Both of the streaming patterns share the property that they are not thread-safe. This means that the neighbouring field access would overwrite data from potential neighbouring threads in a parallel environment. Hence, a second set of PDFs must be allocated. This second grid is used for writing the PDFs. After each time step, a pointer swap is performed, and the algorithms start over.



(a) Pull streaming pattern (b) Push streaming pattern

Figure 2.2: Illustration of the pull (a) and push (b) streaming pattern. The cell of reference is coloured, and the cells are subdivided into nine boxes representing each cell's nine PDFs. Blue populations are read from memory, while red populations are written.

Making the streaming pattern thread-safe and thus inherently parallelisable is a desired property because it allows the removal of the second set of populations and thus cuts the memory requirements in half. This is especially important for execution on general purpose graphics processing units (GPGPUs) where the memory capacity is typically lower than the main memory units of the central processing unit (CPU) and massive parallelism is a requirement. A category of streaming patterns that can achieve this goal is called in-place streaming patterns. Their basic idea is to read and write populations in the same cell. The so-called AA-pattern by Bailey *et al.* [38] fulfils this property by applying a different memory access pattern for even- and odd-numbered time steps. As shown in Figure 2.3a in even-numbered time steps, the collision operation is performed using only cell local information for reading and storing. This is done by writing the updated PDFs in the opposite orientation from their reading position. Contrary to that, the collision operation uses only neighbouring information for reading and writing in odd-numbered time steps, as shown in Figure 2.3b. Another in-place streaming pattern is the so-called

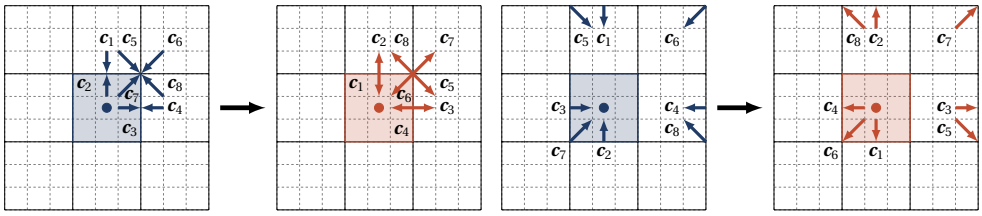


(a) AA-pattern even-numbered time steps (b) AA-pattern odd-numbered time steps

Figure 2.3: The memory access scheme for the AA-pattern streaming algorithm. In even-numbered time steps (a), populations are read locally according to their regular stencil indices. After performing a collision, they are written back to memory and correspond to their inverse stencil indices. In odd-numbered time steps (b), populations are read according to their inverse stencil indices on neighbouring cells. Then, a collision is performed, and the PDFs are stored according to their regular stencil indices on the same neighbouring nodes.

Esoteric Twist of Geier and Schönherr [39]. Similar to the AA-pattern, populations are read

and stored in the same location, making the streaming scheme inherently parallel and thus removing the second set of populations. The idea is to write back PDFs in opposite (twisted) order compared to the reading before the collision. Similarly to the AA-pattern, the Esoteric Twist pattern uses different memory access patterns for even-numbered (compare Figure 2.4a) and odd-numbered (compare Figure 2.4b) time steps. As shown by the work of Geier *et al.* [39], this streaming pattern is especially optimal when indirect addressing is used, i.e., when the memory location of the populations is stored in a second array. The usage of indirect addressing is especially important when large parts of a simulation domain are covered with non-fluid cells. In this case, using a dense continuous array that is directly addressed by looping over it in all directions imposes a non-negligible memory overhead [9].



(a) Esoteric Twist even-numbered time steps (b) Esoteric Twist odd-numbered time steps

Figure 2.4: Memory access scheme for the Esoteric Twist streaming pattern. The switching between regular and inverse stencil indices is similar to the AA-pattern. However, a single collision and a single streaming are performed in each time step. Systematically, the memory locations for reading and writing are determined by looking at the direction each PDF points to. Truncating every negative value in this direction gives their offset, respectively.

Recently, modifications of the Esoteric Twist streaming pattern were proposed by Lehmann *et al.* [40], which are called the Esoteric Pull and the Esoteric Push algorithm. These modifications aim to reduce the number of misaligned stores and improve the performance of the streaming pattern. However, since the Esoteric Twist algorithm was introduced specifically to be suited for indirect addressed lattice Boltzmann (LB) schemes, it is questionable if the Esoteric Pull and the Esoteric Push algorithm keep this advantage. Furthermore, the shift-swap-streaming by Kummerländer *et al.* [41] represents a modification of the AA-pattern by solving the streaming entirely with pointer switches. This streaming pattern falls in the category of the so-called implicit streaming patterns. This means the streaming is not solved directly by addressing the memory of neighbouring cells in each iteration but implicitly by performing pointer manipulations after the collision. The compact streaming by Perepelkina *et al.* [42] involves 2^D cells simultaneously in one streaming cycle. Thus, the collision and streaming are executed on multiple cells instead of one. While this streaming scheme was specifically developed for a temporal blocking algorithm, it must be noted that it was only showcased on toy problems without boundary conditions. Thus, its application to relevant problems is missing.

2.2.2 The Collision Operator

As shown in Section 2.1, the collision process of the PDFs proposes a highly complex part when solving the LBE. The first successful attempts to model the collision were based on the work of Chen *et al.* [19] and Qian *et al.* [35]. In their work, the populations were moved towards a discrete equilibrium $\boldsymbol{f}^{\mathrm{eq}}(\rho, \boldsymbol{u})$ with a particular relaxation time τ. This follows

the idea of the Bhatnagar-Gross-Krook (BGK) formulation as shown in Equation (2.4). The main algorithmic parts are visually represented in Figure 2.5a. While this was a simple and effective approach, it soon became clear that it severely struggles with highly turbulent flow problems where the relaxation time τ becomes smaller and smaller. For this reason, several improvements have been proposed. One category of these improved collision models is based on shifting the collision processes to another function space [21, 33, 36]. The general idea is that the relaxation occurs in a collision space isomorphic to $\mathbb{R}^q$. The equilibrium state for this collision space is given by $\boldsymbol{q}^{\mathrm{eq}}$, and the PDFs are transformed by a bijective mapping $\mathscr{T}$ [1]. Thus, the collision operator can be formulated as:

$$\boldsymbol{q} = \mathscr{T}\left(f\right) \tag{2.12}$$

$$\boldsymbol{f}^{*} = \mathscr{T}^{-1}\left(\boldsymbol{q} + \boldsymbol{S}\left(\boldsymbol{q}^{\mathrm{eq}} - \boldsymbol{q}\right) + \boldsymbol{q}^{\mathrm{F}}\right), \tag{2.13}$$

where $\boldsymbol{f}^{*}$ represents the post-collision populations, $\boldsymbol{S} = \mathrm{diag}(\omega_0, \ldots, \omega_q)$ the relaxation matrix (with $\omega_n = 1/\tau_n$) and $\boldsymbol{q}^{\mathrm{F}}$ a force term that represents external forces, e.g., caused by gravitational acceleration on the PDFs. This source term especially plays an important role when modelling other PDEs than the NSEs, as we will show in Chapter 3.

The first collision space proposed in the literature was based on shifting populations to a basis formed by the moments of the PDFs [43, 44]. This collision space is often referred to as moment space, and its first versions can be understood as a simple generalisation of the BGK collision operator. This brings a main advantage: each moment can be relaxed with a different relaxation time towards its equilibrium state. Hence, its name multiple-relaxation-time (MRT) collision operator. In this fashion, it is possible to improve the accuracy and stability of the method, and it becomes possible to separate shear and bulk viscosity [45]. Generate functions are necessary to introduce and analyse moment-based collision operators properly. For raw-moments, a moment-generating function M of the PDF f is defined as

$$M\left(\Xi\right) = \int \exp\left(\xi\Xi\right) f\left(\xi\right) d\xi. \tag{2.14}$$

Thus, the moment-generating functions represent a multidimensional Laplace transformation of the PDFs. To ensure that f is integrable, the Dirac delta function is applied to the distribution as $f\left(\boldsymbol{c}\right) \coloneqq \sum_i f_i \delta\left(\boldsymbol{c} - \boldsymbol{c}_i\right)$. Now, to derive the monomial raw moments $m_{\alpha\beta\gamma}$ of f, a mixed derivative is applied to the moment-generating function M. The mixed derivative is then evaluated at zero:

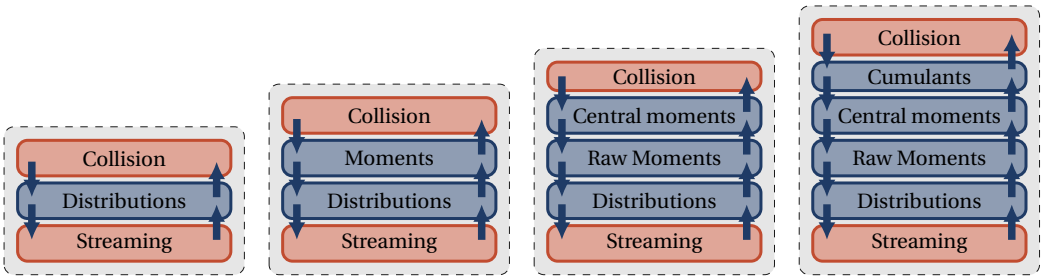
$$m_{\alpha\beta\gamma} \coloneqq \partial_1^{\alpha}\partial_2^{\beta}\partial_3^{\gamma} M\left(\Xi\right)\Big|_{\boldsymbol{\Xi}=\boldsymbol{0}}, \tag{2.15}$$

where monomial quantities are defined by a monomial $x^{\alpha}y^{\beta}z^{\gamma}$ as $q_{\alpha\beta\gamma}$. Thus, a specific type of quantities defines the *basis* of the collision space, where the basis is a sequence $\left(p_i\right)_{i=0,\ldots,q-1}$ of linear independent polynomials. Through a linear combination of monomial moments, it is possible to obtain polynomial moments as [1]

$$m_{\alpha\beta\gamma} = \sum_i f_i c_{i,x}^{\alpha} c_{i,y}^{\beta} c_{i,z}^{\gamma}, \quad m_p = \sum_i f_i\, p(\boldsymbol{c}_i). \tag{2.16}$$

The transformation to a moment space with basis $\left(p_i\right)_i$ is typically represented by a linear invertible matrix $\boldsymbol{M}$, called moment matrix [33, 44]. In summary, the collision in the moment space is done by transforming the populations to the moment space, applying the collision in the moment space, transforming the populations back and applying

the streaming step. This is shown in Figure 2.5b. It is important to note that the BGK operator can be recovered by using the monomial moments as the basis and applying the same relaxation rate to each moment. In this case, $\boldsymbol{S}$ becomes a scalar matrix with the single relaxation rate ω on its diagonal. Since the same relaxation rate is used for all moments, this method is also referred to as single-relaxation-time (SRT) collision operator. Using a single relaxation time limits the number of free parameters. However, moments represent physical quantities which might relax towards their equilibrium at different speeds. Due to this fact, it is also desired that moments are statistically independent to relax them independently of each other to their equilibrium state. Therefore, it is common to choose the moments mutually orthogonal on the discrete velocity set $\boldsymbol{c}_i$ with a uniform weight function [44, 46]. This mutual orthogonality is commonly achieved by applying a Gram–Schmidt process [47] to the raw moments. Unfortunately, this leads to undesired coupling between the conserved quantities, i.e., flow density ρ and velocity $\boldsymbol{u}$, and some higher order moments that are not linked to hydrodynamics [48, 49]. These undesired couplings can degenerate the collision operator's stability and accuracy. A non-uniform weight function is introduced to remove it, resulting in a so-called weighted multiple-relaxation-time (WMRT) model [50].



(a) BGK (b) MRT (c) CMRT (d) Cumulants

Figure 2.5: Overview of different moment-based collision models. Classically, the streaming step is based on the PDFs while the collision step is executed using PDFs (a), moments (b), central-moments (c) or cumulants (d). Several transformation steps must be performed depending on the statistical quantity of choice.

Despite the moment-based collision operator's increased stability, it was still impossible to reliably simulate highly turbulent flow phenomena [36]. Thus, shifting the collision processes to the co-moving frame of reference was further suggested. This means that the statistical observer moves with the macroscopic velocity of the flow field [36, 51, 52]. Additionally, shifting the frame of reference repairs violations of Galilean invariance [51]. This velocity shift is reflected in the Laplace frequency space with $\exp(-\Xi \cdot \boldsymbol{u})$. Thus, the central moment-generating function can be defined as

$$K(\Xi) := \exp(-\Xi \cdot \boldsymbol{u})\, M(\Xi). \tag{2.17}$$

Similarly, the derivatives of the central moment-generating function give the monomial central moments with respect to the velocity shift

$$\kappa_{\alpha\beta\gamma} = \sum_i f_i (\xi_{i,x} - u_x)^\alpha (\xi_{i,y} - u_y)^\beta (\xi_{i,z} - u_z)^\gamma, \quad \kappa_p = \sum_i f_i\, p(\boldsymbol{\xi}_i - \boldsymbol{u}). \tag{2.18}$$

The transformation to the central moment space can be expressed with a linear transformation matrix $\boldsymbol{K}$. However, the matrix $\boldsymbol{K}$ is generally dense, and thus often, the shift to the central moment-space is separated by a shift to the moment space with $\boldsymbol{M}$ followed by a second transformation with the so-called shift-matrix $\boldsymbol{N}$ [52–54]. Thus, in Figure 2.5c, the collision operator is pictured with two transformation steps. Formulating the linear transformations as matrices forms an easy-to-understand algorithm; however, more efficient methods have been introduced based on recursive formulations via temporal quantities [1, 55]. A detailed description of these advanced methods is given in Section 2.2.2. While central-moments improve the Galilean invariance, they are not statistically independent quantities [49]. This was corrected by using cumulants as observers. The cumulant-generating function is defined using the natural logarithm

$$C(\Xi) := \log M(\Xi), \quad c_{\alpha\beta\gamma} := \partial_1^\alpha \partial_2^\beta \partial_3^\gamma C(\Xi)\Big|_{\Xi=\mathbf{0}}, \quad C_{\alpha\beta\gamma} := \rho c_{\alpha\beta\gamma}. \tag{2.19}$$

The monomial cumulants can be obtained by an evaluation of the mixed derivatives of the cumulant-generating function at the origin. For the rescaled cumulants, $C_{\alpha\beta\gamma}$, i.e., the polynomial cumulants, Geier *et al.* [49] is taken as a reference. The transformation to the cumulant space is not linear and, thus, generally complex. Therefore, populations are first shifted to the central-moment space to simplify this transformation. This is because some central moments and cumulants are already equal; thus, shifting these central moments is unnecessary. An overview of the transformation process is shown in Figure 2.5d.

Ever since the introduction of the MRT collision operator, the number of free parameters for the LBM strongly increased [44]. The simulation scenario often fixes only the relaxation time corresponding to the fluid's viscosity. In contrast, relaxation times for higher-order moments can be chosen freely in the stability window as these are not physical but algorithmic parameters [33]. In the beginning, this was stated as a great advantage because it allows tuning the stability and accuracy of the MRT model. However, numerical analysis usually reveals that optimal choices of the higher order relaxation times only exist for specific cases, and general values can't be provided [2, 56]. This problem opened the door for many modifications of the presented collision processes that should be discussed briefly in the following:

Two-Relaxation-Time Model

Ginzburg *et al.* [57, 58] proposed a prominent approach for choosing the missing free parameters. In their work, the idea was to divide the moments into even and odd moments. Thus, only one free parameter remains because the even moments are relaxed by the relaxation time derived from the fluid's viscosity (ω^+). The relaxation time for the odd moments (ω^-) is calculated from the relaxation time of the even moments using the so-called *magic parameter*

$$\Lambda = \left(\frac{1}{\omega^+ \Delta t} - \frac{1}{2}\right)\left(\frac{1}{\omega^- \Delta t} - \frac{1}{2}\right). \tag{2.20}$$

The magic parameter is not chosen arbitrarily, but mathematical analysis reveals optimal parameters for stability and accuracy in specific cases. However, one of the weaknesses of this approach is that the choice of the magic parameter is difficult in more complex cases. Due to the usage of two relaxation times, this model is often called the two-relaxation-time (TRT) collision operator.

Parametrisation of Cumulants

Similar to the work of Ginzburg *et al.* [57, 58], Geier *et al.* [59, 60] looked for optimal choices for the relaxation rates. In this case, however, the cumulants formed the basis of the analysis. As stated before, cumulants are statistically independent quantities. Thus, relaxation rates can be defined and analysed independently. Applying Taylor expansion and asymptotic analysis, Geier *et al.* found:

$$\omega_3 = \frac{8\left(2\omega_1^2 - 3\omega_1 - 2\right)}{7\omega_1^2 - 14\omega_1 - 8} \tag{2.21}$$

$$\omega_4 = \frac{8\left(\omega_1 - 2\right)\left(4\omega_1 - 7\right)}{9\omega_1^2 - 50\omega_1 + 56} \tag{2.22}$$

$$\omega_5 = \frac{24\left(3\omega_1^3 - 13\omega_1^2 + 12\omega_1 + 4\right)}{29\omega_1^3 - 130\omega_1^2 + 152\omega_1 + 48} \tag{2.23}$$

$$A = \frac{-3\omega_1^2 + 2\omega_1 + 4}{5\omega_1^2 - 7\omega_1 + 2} \tag{2.24}$$

$$B = \frac{2\left(-7\omega_1^2 + 14\omega_1 + 2\right)}{3\left(5\omega_1^2 - 7\omega_1 + 2\right)}. \tag{2.25}$$

With this solution, relaxation rates for cumulants up to third order have been found. Higher order relaxation rates are unnecessary because they do not influence the leading error, and thus, they can be set to 1. Besides the relaxation rates, two additional values, A and B, emerge, influencing third-order correction terms applied to the cumulants. Importantly to note here is that Equations (2.21) to (2.25) are stated in the limit of $\omega_2 = 1$, where ω_2 influences the bulk viscosity. With these relations, it could be shown that the resulting collision operator becomes fourth-order accurate with respect to the equivalent diffusion term of the NSE [60, 61]. However, these optimisations on the accuracy come with a degeneration of the stability of the model, which can be fixed by applying limiters to the functions [55, 60, 61]. Similar to the literature, this method will be called the K17 cumulant method in the remainder of this thesis.

Regularisation

Another important idea to improve the stability of the collision operator is to filter out non-hydrodynamic contributions by applying regularisations as originally shown by Latt *et al.* [62]. Their original work applied the regularisation to the SRT collision operator. However, later work shows that regularisation translates to setting non-hydrodynamic relaxation rates to unity [36]. Thus, regularisation can be applied to all categories of collision operators. In general, the goal is to increase the stability of a collision model. Unfortunately, it could be shown that this can violate the model's accuracy. In fact, Geier *et al.* [60] showed that it was impossible to capture the drag crisis of a spherical object using the regularised cumulant collision operator.

Further extensions of the original regularised collision model are done by applying a recursive update rule based on Hermite polynomials [63]. This idea comes from the realisation that the original regularisation process can not completely filter out non-hydrodynamic contributions as Latt *et al.* had hoped. Applying a blending function to the second-order Hermite polynomials leads to the hybrid recursive regularised collision operator [64]. The blending function increases the stability while also controlling the

dissipation of the model with an additional parameter. However, it comes with the downside that the blending function needs the velocity gradient, which needs to be calculated from the macroscopic velocity field. Thus, it degenerates the collision operator's locality. Comparisons between the hybrid recursive regularised and the K17 collision models were made by Spinelli *et al.* [65].

Entropic Stabilisation

Another possibility to improve the stability of the collision operator is to enforce the validity of the H theorem after the discretisation of the Boltzmann equation [66, 67]. This leads to a minimisation problem solved at each grid point at each time step. Initially, this idea was applied to SRT collision operators. However, in this case, it inevitably leads to a non-constant dynamic viscosity because the minimisation problem is applied to the relaxation rate of the collision operator. While this was initially seen as problematic, it was later shown that the method falls in the same category as large eddy simulation (LES) turbulent models [68]. With the increasing importance of MRT based collision methods, later versions of entropic stabilisation were also developed to modify higher-order relaxation rates exclusively. These methods are often called KBC collision models [69, 70].

2.2.3 Boundary Conditions

One of the biggest strengths of the LBM is that it's typically applied to regular Cartesian grids [33]. This removes a tedious meshing process and simplifies simulations involving complex geometries, like flow through porous media. On top of that, standard boundary conditions, e.g., to impose no-slip conditions, are especially simple to formulate in the common halfway bounce-back rule [33]

$$f_{\bar{i}}\left(\boldsymbol{x}_b, t+\Delta t\right) = f_i\left(\boldsymbol{x}_\mathrm{F}, t\right) - 2 w_i \rho_w \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}_w}{c_s^2}. \tag{2.26}$$

Here $f_{\bar{i}}$ refers to f for the direction $\bar{i}$ such that $\boldsymbol{c}_{\bar{i}} = \boldsymbol{c}_i$. Furthermore, w_i describes the lattice weights, and $c_s^2 = {}^{\Delta x}/_{\sqrt{3}\Delta t}$ is the lattice speed of sound. A no-slip boundary condition imposes that a viscous fluid attains zero bulk velocity at a solid boundary. For this case $\boldsymbol{u}_w = \boldsymbol{0}$. However, it is also possible to simulate moving walls, also referred to as velocity bounce-back (UBB) boundary conditions, by imposing a certain value for $\boldsymbol{u}_w$. It is important to note that the density at the boundary ρ_w can not be prescribed by the bounce-back scheme. Rather, it must be estimated. Commonly, this is done by simply using the density from the first fluid node at the wall, but also other ideas have emerged [33]. The halfway bounce-back boundary condition is illustrated in Figure 2.6.

For grid-aligned Poiseuille profiles, it could be shown that this boundary condition can be made third-order accurate [71, 72]. However, when applied to complex geometries, the solution generally degenerates to a stair-case approximation and thus falls back to first-order accuracy [73, 74]. This poses a severe problem and reduces the general accuracy of the simulation. Thus, the first claimed strength of simple boundary treatment quickly becomes a disadvantage when not treated carefully. An attempt to solve this problem was made by Bouzidi *et al.* [75]

$$f_{\bar{i}}(\boldsymbol{x}_b, t+\Delta t) = \begin{cases} \frac{1}{2q} f_i(\boldsymbol{x}_\mathrm{F}, t) + \frac{2q-1}{2q} f_{\bar{i}}(\boldsymbol{x}_\mathrm{F}, t), & \text{if } q \geq 0.5 \\ 2q f_i(\boldsymbol{x}_\mathrm{F}, t) + (1-2q) f_i(\boldsymbol{x}_\mathrm{FF}, t), & q > 0 \wedge q < 0.5 \end{cases}. \tag{2.27}$$

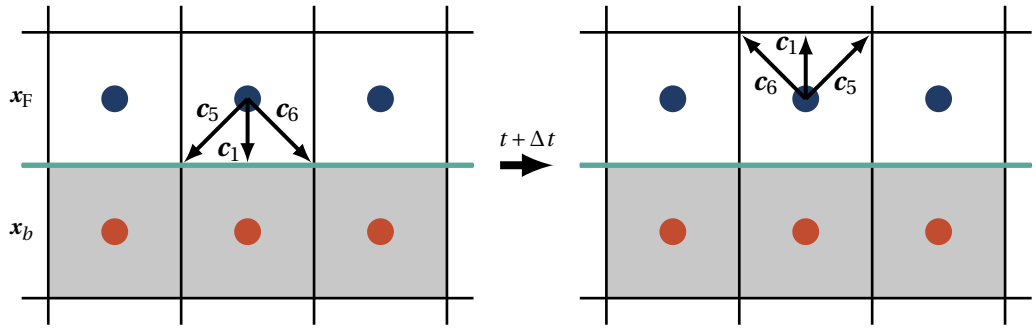


Figure 2.6: Visual representation of the halfway bounce-back boundary condition. Blue nodes mark fluid nodes, while red nodes show boundary nodes, i.e., nodes inside the boundary or outside the domain. The boundary is visualised in green. After a time step, lattice directions pointing towards the boundary end up in opposite directions to the wall.

The idea here is to use a second fluid node at $\boldsymbol{x}_{FF}$ and use it to apply an interpolation depending on the real distance to wall q. An illustration in a two-dimensional setup is shown in Figure 2.7, where an arbitrarily complex boundary is drawn in green. Important to note here is that the wall distance q is not uniform per cell but must be calculated per lattice direction. In the shown illustration, e.g., both cases covered by Equation (2.27) appear in the same cell for different lattice directions. While this boundary condition shows an improvement by recovering the second-order convergence of the LBM, it comes with the undesired side effect that a second fluid node is needed and, thus, with an increased stencil.

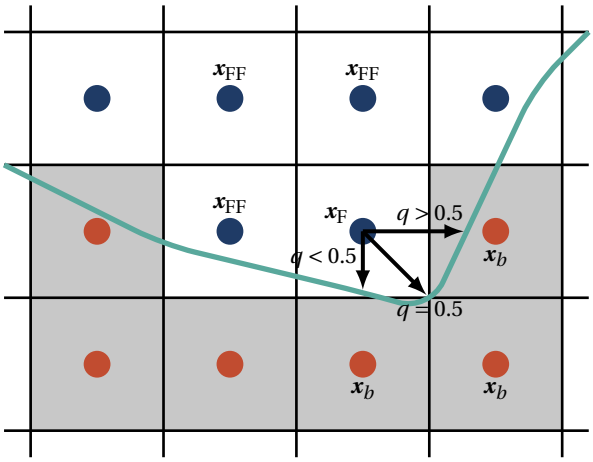


Figure 2.7: Interpolated bounce-back boundary condition depending on the wall distance $q \in [0, 1]$. Fluid nodes are marked in blue, while red points represent solid boundary nodes. An arbitrary boundary is pictured as the green line through the domain. Complex shapes generally need interpolation to preserve the simulation's second-order accuracy.

This problem was solved by the work of Geier *et al.* [49]. Using post-collision PDFs, a cell local interpolation scheme is possible. In contrast to Bouzidi's bounce-back boundary condition, the interpolation is not performed using more information spatially but rather extending information temporarily. Thus, besides the post-collision populations, pre-collision populations are also involved. To avoid storing the missing pre-collision PDF, a

back calculation based on the BGK rule can be performed:

$$f_{\bar{i}}^{\mathrm{p}}(\boldsymbol{x}_{\mathrm{F}},t)=\frac{f_{\bar{i}}(\boldsymbol{x}_{\mathrm{F}},t)-f_i(\boldsymbol{x}_{\mathrm{F}},t)}{2}+\frac{f_{\bar{i}}(\boldsymbol{x}_{\mathrm{F}},t)+f_i(\boldsymbol{x}_{\mathrm{F}},t)-\omega\left(f_{\bar{i}}^{\mathrm{eq}}(\rho,\boldsymbol{u})+f_i^{\mathrm{eq}}(\rho,\boldsymbol{u})\right)}{2-2\omega} \tag{2.28}$$

$$f_{\bar{i}}^{\mathrm{wall}}(\boldsymbol{x}_{\mathrm{F}},t)=(1-q)f_{\bar{i}}^{\mathrm{p}}(\boldsymbol{x}_{\mathrm{F}},t)+qf_{\bar{i}}(\boldsymbol{x}_F,t) \tag{2.29}$$

$$f_{\bar{i}}(\boldsymbol{x}_b,t+\Delta t)=\frac{1}{q+1}f_{\bar{i}}^{\mathrm{wall}}(\boldsymbol{x}_{\mathrm{F}},t)+\frac{q}{q+1}f_i(\boldsymbol{x}_{\mathrm{F}},t). \tag{2.30}$$

It is important to note that Equation (2.27) and Equation (2.28) can be modified to impose a velocity at the wall similarly to Equation (2.26). Furthermore, imposing high-order boundary conditions is a vast field of research in the lattice Boltzmann community. Thus, besides the boundary conditions here, many more complex interpolation schemes have been developed over the past decades. A good overview can be found in the work of Francesco Marson [76].

To impose the density at open boundaries, the anti-bounce-back boundary condition can be used [77]

$$f_{\bar{i}}(\boldsymbol{x}_b,t+\Delta t)=-f_i(\boldsymbol{x}_{\mathrm{F}},t)+2f_i^{\mathrm{eq}}(\rho_w,\boldsymbol{u}). \tag{2.31}$$

Since the access pattern of the populations remains similar to the UBB boundary conditions, however, flipped in the sign, these boundary conditions are named anti-bounce-back. For this family of boundary conditions, the equilibrium distribution is calculated using the imposed density and a missing velocity $\boldsymbol{u}$. The velocity at the first fluid node is commonly used for the missing velocity. However, this leads to problems, especially for simulations featuring high velocities. Furthermore, it is important to note that with the state equation $p=c_s^2\rho$, anti-bounce-back boundary conditions can prescribe a pressure outlet at the domain [33].

Due to strong acoustic reflections caused by anti-bounce-back boundary conditions Geier *et al.* [49] proposed an alternative outlet boundary

$$f_{\bar{i}}(\boldsymbol{x}_b,t+\Delta t)=f_{\bar{i}}(\boldsymbol{x}_b+\Delta x,t-\Delta t)(c_s-\boldsymbol{u})\frac{\Delta t}{\Delta x}+\left(1-(c_s-\boldsymbol{u})\frac{\Delta t}{\Delta x}\right)f_{\bar{i}}(\boldsymbol{x}_b+\Delta x,t-\Delta t). \tag{2.32}$$

The idea of this boundary condition is to copy populations from the location in space and time where pressure waves are coming from. Hence, the boundary condition needs information in the normal direction of the outlet.

The last family of boundary conditions important within this work are free-slip boundary conditions. Unlike no-slip boundaries, free-slip conditions do not impose any restriction on the tangential velocity of the flow at the wall. Thus, they can be seen as the opposite of the no-slip conditions, which impose a zero velocity at the wall. Free-slip boundary conditions are also important because they can be used as symmetry conditions. Thus, in special setups, it is possible to obtain all flow characteristics from just half the simulation domain [33]. To apply free-slip conditions on the LBM populations pointing towards the boundary, PDFs get reflected in their mirror direction:

$$f_{\bar{i}}\left(\boldsymbol{x}_b+\boldsymbol{c}_j,t+\Delta t\right)=f_i(\boldsymbol{x}_{\mathrm{F}},t), \tag{2.33}$$

where $\boldsymbol{c}_j$ is the tangential discrete velocity [78]. An illustration of free-slip boundary conditions can be found in Figure 2.8.

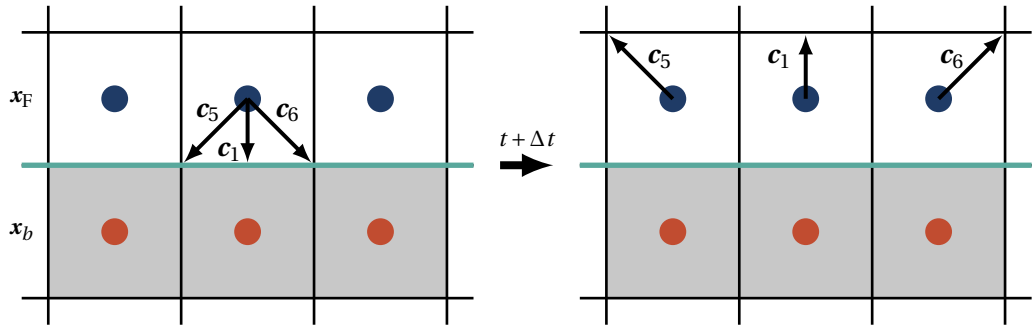


Figure 2.8: Visual representation of free-slip boundary condition. Blue nodes mark fluid nodes, while red nodes show boundary nodes, i.e., nodes inside the boundary or outside the domain. The boundary is visualised in green. After a time step, the lattice directions pointing towards the boundary end up in tangential directions opposite to the wall.

2.2.4 External Forces

Introducing force terms to the LBE is crucial in many applications. One of the most prominent cases is to express a physical force density on the fluid, for example, in the form of a gravitational acceleration $\boldsymbol{\alpha}$ [33]:

$$\boldsymbol{F}_\mathrm{g} = \rho \boldsymbol{\alpha}. \tag{2.34}$$

However, force terms are also crucial in modelling additional physical phenomena to the LBE or to correct truncation errors. An important example in this work is the modelling of multiphase flows as shown in Chapter 3. When combining physical forces with the LBE, the problem arises with mapping the D-dimensional force vector to the Q-dimensional population space. A simple way to do so is by

$$\boldsymbol{q}_\mathrm{S}^\mathrm{F} = \mathcal{T}\left(\frac{w_i}{c_s^2}\boldsymbol{c}_i \cdot \boldsymbol{F}_\mathrm{g}\right). \tag{2.35}$$

In Equation (2.35), several important properties of the force vector $\boldsymbol{q}^\mathrm{F}$ are illustrated. The first important property is that its contribution to the density is zero. Thus, this force transformation scheme ensures that the force stays a momentum source, not a mass source. Hence, the force should act with its full contribution to the momentum moments. Lastly, it is important to note that generally, force terms are applied in the respective collision space as indicated by the multiplication with the bijective mapping $\mathcal{T}$. The last point is not necessary. However, it was shown to give computational advantages, e.g., in the thesis of Travis Mitchell [79].

Another important transformation model was proposed by the work of Guo *et al.* [80]:

$$\boldsymbol{q}_\mathrm{Guo}^\mathrm{F} = \left(\boldsymbol{I} - \frac{1}{2}\boldsymbol{S}\right)\mathcal{T}\left(w_i \cdot \left(\frac{\boldsymbol{c}_i - \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{c}_i - \boldsymbol{u})\,\boldsymbol{c}_i}{c_s^4}\right) \cdot \boldsymbol{F}_\mathrm{g}\right). \tag{2.36}$$

The idea is to apply a symmetric forcing scheme to the collision process in the LBE. This means applying half of the force term before and the other half after the collision. Thus, the relaxation matrix $\boldsymbol{S}$ appears with the factor of a half. Therefore, the calculation of macroscopic quantities also needs to be adapted. The adapted calculation of the velocity reads:

$$\boldsymbol{u} := \frac{1}{\rho}\sum_{i=0}^{q-1} f_i \boldsymbol{c}_i + \frac{\Delta t}{2}\boldsymbol{q}_i^\mathrm{F}. \tag{2.37}$$

Contrary to Equation (2.35), Equation (2.36) possesses contributions to higher-order moments. While there is a wide variety of different force transformations stated in the literature, usually, these aim to filter special contributions in specific applications. A good overview can be found in references [33, 81, 82].

2.3 Complete Lattice Boltzmann Algorithm

The complete algorithm to simulate single-phase flows with the LBM is given by Algorithm 1. The algorithm begins by initialising the PDFs. A simple strategy to initialise the PDFs is to assign them their equilibrium values

$$\boldsymbol{f}(\boldsymbol{x}, t=0) = \boldsymbol{f}^{\mathrm{eq}}\left(\rho(\boldsymbol{x}, t=0), \boldsymbol{u}(\boldsymbol{x}, t=0)\right). \tag{2.38}$$

This approach causes errors at the initialisation when gradients exist in the initial velocity or density field [83]. In the cases here, however, the long-term behaviour of the simulations is analysed. These cases are relatively insensitive to the initial condition [33]. Therefore, it is assumed to be sufficient here. Furthermore, the simulations conducted in this work generally start at rest.

Afterwards, boundary conditions from the domain boundaries or obstacles are mapped to a flag field. This flag field is used to set up index lists to efficiently execute the boundary conditions, while the main compute kernel can be executed without any branches. More details are provided in Section 4.4.5. The main compute kernel is the combined stream-collide method, which follows Equation (2.6). It is executed in the timestep loop along with the boundary conditions and synchronisation kernels. The synchronisation here refers to the data exchange between subdomains[1] of the simulation region. A detailed description of this data synchronisation is given in Section 4.4.4.

Algorithm 1: Single phase fluid simulation with the LBM

```
1 Initialise PDFs
2 Initialise flag field with bounding walls and obstacles (Section 4.4.5)
3 for each time step t do
4 |   Perform stream-collide in each cell (Sections 2.2.1 and 2.2.2)
5 |   Apply boundary conditions (Section 2.2.3)
6 |   Synchronise PDFs (Section 4.4.4)
7 end
```

[1] In WALBERLA these subdomains are the blocks in the block-structured grid

CHAPTER 3

EXTENSION TO MULTIPHASE FLOWS

Contents

After introducing the basic principles of the lattice Boltzmann method, an extension to multiphase flow problems is shown in this chapter. This is done by first setting a background for multiphase flow problems and various models developed for the lattice Boltzmann method. Among these models, a recent approach is based on solving the conservative Allen-Cahn equation and an incompressible lattice Boltzmann based fluid solver. As this model forms a powerful tool for high Reynolds number multiphase problems with high-density ratios, it will be introduced in detail, and later in this thesis, the model forms the basis of thorough numerical studies. The chapter ends by extending the conservative Allen-Cahn model to thermocapillary flow problems. In these problems, the surface tension is temperature dependent; thus, a temperature solver must be coupled. This opens the door for microfluidics problems that occur in countless natural phenomena and industrial applications.

3.1 Background

Multiphase flows refer to computational fluid dynamics (CFD) problems with multiple fluids of different physical characteristics. Most prominently, they differ in density and/or viscosity. However, other quantities, like thermal conductivity, can also differ in multiphase flow problems. The most important distinction of multiphase flows is between single and multicomponent flows. Single-component flow problems are characterised by a mixture of one fluid in different states.

A typical example would be a mixture of liquid water and water vapour. Thus, the two phases can convert to each other. For example, boiling liquid water will eventually convert to water vapour. Conversely, miscible or immiscible multicomponent flows consist of different non-convertible fluids like water and oil [33].

Similar to the large variety of different multiphase problems, many techniques exist to capture their behaviour numerically. As the system consists of fluids with different properties, one of the straightforward ideas would be to model each fluid with its own solver for the Navier-Stokes equation (NSE). Such approaches are called sharp interface approaches because each of these solvers evolves on its own numerical mesh, and thus, the interface between the fluids is virtually non-existent. However, as a consequence, the interface acts as a moving boundary condition on the solvers and, thus, can be difficult to model [84–88]. On the other side, if one of the fluids in the system plays only a secondary importance in the simulation analysis, it is possible only to model the moving boundary and leave out the evolution of that fluid on the mesh entirely. This gives rise to the idea of free-surface simulations [89].

On the other hand, diffuse interface models follow a different path. The idea is to solve the entire simulation domain with one consistent model and distinguish between different fluid phases by introducing a so-called order parameter

$$\phi(\boldsymbol{x}) = \begin{cases} \phi_{\mathrm{L}} & \text{if } \boldsymbol{x} \in \text{fluid A} \\ \phi_{\mathrm{H}} & \text{if } \boldsymbol{x} \in \text{fluid B.} \end{cases} \tag{3.1}$$

Near the interface between the fluids, a function $I(\boldsymbol{x}, W)$ is employed that provides a smooth transition in the order parameter. In diffusive interface models, the interface typically governs a certain thickness W that also influences the steepness of the smooth function. Common formulations for it are based on higher order polynomials or trigonometric functions [90]. It is important to note that the fluids can be marked with arbitrary values for the order parameter. In this section, the values 0 and 1 are chosen as an example.

With the order parameter ϕ, it is then possible to determine the characteristic properties of each fluid at each point in the domain exactly. For example the density ρ and the kinematic viscosity ν can be recovered by,

$$\rho(\boldsymbol{x}) = \rho_{\mathrm{L}} + \left(\rho_{\mathrm{H}} - \rho_{\mathrm{L}}\right)\phi(\boldsymbol{x}), \tag{3.2}$$

$$\nu(\boldsymbol{x}) = \nu_{\mathrm{L}} + (\nu_{\mathrm{H}} - \nu_{\mathrm{L}})\,\phi(\boldsymbol{x}). \tag{3.3}$$

Modelling the interface between the fluids with a certain thickness comes with the downside that it typically requires more memory than sharp interface approaches because the problem must be resolved sufficiently to keep the interface sharp enough compared to the bulk. However, the big advantage of this approach is that the whole simulation domain is treated with the same update rule, and no distinction needs to be made. This typically results in simpler algorithms because no complex re-meshing or algorithmic treatment of special cases needs to be taken care of [3, 91].

3.2 The Lattice Boltzmann Method for Multiphase Flows

As shown in Section 2.2, the lattice Boltzmann method (LBM) offers many advantages over conventional CFD solvers. Due to the explicit nature of the LBM, it forms a natural candidate for unsteady problems, which is typical for multiphase flow problems. Furthermore, due to its origin directly from the Boltzmann transport equation (BTE) it allows the method more control over molecular interactions [79]. Several models exist in the lattice Boltzmann community to recover multiphase systems' behaviour [92].

3.2.1 Colour Gradient

Among the most important multiphase models for the LBM and also the earliest one is the so-called colour gradient method developed by Gunstensen *et al.* [93]. As the name suggests, using different colours, the colour gradient method identifies the components in a two-component flow with individual colours. It is important to note that the model is restricted to two-component flows; thus, two-phase flows are inaccessible. The idea is then to use three particle distribution function (PDF)s. One set of populations refers to the so-called colourless population, while the others indicate the defined colours. The three populations enter an extended collision phase

$$\boldsymbol{f}^{*} = \boldsymbol{f} + \boldsymbol{\Omega} + \boldsymbol{\Omega}^{\mathrm{p}}, \tag{3.4}$$

where $\boldsymbol{\Omega}$ refers to a normal collision operator as shown in Section 2.2.2, while $\boldsymbol{\Omega}^{\mathrm{p}}$ points to a second term that is referred to as a perturbation term. The perturbation term allows the introduction of surface tension and can be stated as,

$$\Omega_i^{\mathrm{p}} = A|\boldsymbol{C}|\cos(2\theta_i). \tag{3.5}$$

The control over the surface tension is then given through the parameter A. Furthermore, $\boldsymbol{C}$ is the so-called colour gradient and thus forms the basis of the method. The colour gradient is described via an order parameter that comes from the density of the individual fluids,

$$\phi(\boldsymbol{x}, t) = \rho_r(\boldsymbol{x}, t) - \rho_b(\boldsymbol{x}, t) \tag{3.6}$$

$$\boldsymbol{C}(\boldsymbol{x}, t) = \sum_i \boldsymbol{c}_i \phi(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t). \tag{3.7}$$

The angle between the lattice populations and the colour gradient is defined by θ. The perturbation term can be understood as re-orientating the flow parallel to the interface between the individual fluids. However, a second step needs to be executed to allow a separation of phases. This second step is called the "recolouring" operator and updates the two population sets responsible for the colouring. Especially the nonphysical background of the recolouring step caused problems at the early stages of the method. However, later advancements extended the model's applicability strongly [33, 92, 94–96].

3.2.2 Pseudo Potential

Another popular approach to simulate multiphase flows with the LBM was proposed by Shan and Chen [97, 98]. Their idea falls in the category of so-called pseudo potential models, where the ideal gas equation of state from the single phase LBM is replaced by a non-ideal non-monotonic equation of state [92, 97]. This allows for the simulation of both single- and multi-component problems. However, additional PDFs need to be introduced for multi-component flows, while single-component flows can be simulated using just a single set of PDFs. The thought behind the model is that repulsive forces must act between the molecules of different phases in the fluid mixture. This inter-particle force is commonly given by

$$\boldsymbol{F}_p(\boldsymbol{x},t) = -\boldsymbol{\Psi}(\boldsymbol{x},t)\sum_i G w_i \boldsymbol{c}_i \boldsymbol{\Psi}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t,t) \tag{3.8}$$

$$\boldsymbol{F}_p^\sigma(\boldsymbol{x},t) = -\boldsymbol{\Psi}^\sigma(\boldsymbol{x},t)\sum_{\tilde{\sigma}:=\sigma} G_{\sigma\tilde{\sigma}}\sum_i w_i \boldsymbol{c}_i \boldsymbol{\Psi}^{\tilde{\sigma}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t,t) \tag{3.9}$$

for single- and multi-component problems, respectively. The core part of these force functions is contained in the kernel function G, which controls the strength of the force and the pseudopotential $\boldsymbol{\Psi}$. It replaced the actual density ρ with an effective density. The reasoning behind this is mostly of a numerical nature. Originally, the pseudopotential was given as,

$$\Psi(\rho) = \rho_0\left(1-\exp\left(\frac{-\rho}{\rho_0}\right)\right). \tag{3.10}$$

The beauty of the pseudopotential model lies in its simplicity. The only change to the LBM is the additional forcing term, which can be added as shown in Section 2.2.4. Even in the cases of multi-component flows, the individual components σ are just coupled by a rather simple force sum over the components. Theoretically, this even allows the simulations of infinitely many components in this simple fashion [33]. However, the simplicity also comes with its drawbacks. One severe limitation of the model is that all parameters are tightly coupled. In practice, this means that the interface width and the surface tension force can not be controlled individually. Thus, applications requiring this level of control are hard to access with the model. Improvements in this regard have been made, for example, by Lycett-Brown *et al.* [99]. Furthermore, applying a Taylor expansion to Equation (3.10) reveals an undesired inconsistency with thermodynamics encoded within the model. The inconsistency can be explained by missing long-range particle interactions occurring on the molecular basis, which the model arises from [33, 92, 100]. Due to its natural combination with the LBM it oftentimes forms a good entry to study multiphase flows with the LBM [33, 92, 101].

3.2.3 Free-Energy

To develop a thermodynamically consistent model, Swift *et al.* [102, 103] followed the idea of a free-energy approach. As the name already indicates, the roots of this approach directly lie in the thermodynamic laws underlining the motion of the fluid. Thus, by design, thermodynamic inconsistencies can not occur in these kinds of models. To account for multiphase flows, adaptions to the macroscopic equations have to be made,

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}\right)=-\nabla p+\nabla\cdot\left(\mu\left[\nabla\boldsymbol{u}+(\nabla\boldsymbol{u})^{T}\right]\right)+\boldsymbol{F}_{S}+\boldsymbol{F}_{b}. \tag{3.11}$$

Compared to Equations (2.7) and (2.8) the momentum equation has to be adapted. In homogeneous isotropic fluids, the pressure is equal in all directions. Consequently, the pressure can be described as a scalar value. This no longer holds in multiphase systems, which introduces the necessity to describe ∇p by a pressure tensor $P_{\alpha\beta}$. The surface tension that arises at the interface of two fluids is introduced as a forcing term $\boldsymbol{F}_S$ to Equation (3.11). This is another key difference between Equation (3.11) and Equation (2.8). Sometimes, the surface tension force term is incorporated within the pressure tensor.

The additional information needed now can be obtained from the free energy functional. This functional can be freely chosen and commonly targets the fluids' density or order parameter in multi-component systems. A general form of such a free energy functional could be given in this form

$$\boldsymbol{\Psi}=\int_{V}\left[\Psi_{b}(\boldsymbol{x},t)+\Psi_{g}(\boldsymbol{x},t)\right]dV+\int_{A}\Psi_{s}(\boldsymbol{x},t)\,dA, \tag{3.12}$$

where the free energy at the bulk is given by Ψ_b. Most importantly, this term must modify the equation of state to allow phase separation. Furthermore, Ψ_g can be understood as a penalty term for gradients in the order parameter. Thus, this term minimises the interface surface area and is strongly related to the surface tension of the fluids. The last term Ψ_s comes into play for solid interaction, thus it provides the necessary information to impose specific wetting conditions [33, 92]. Free energy formulations based on the LBM have initially struggled with wide adoption because the original model suffered from a violation of the Galilean invariance. To tackle this problem, Holdych *et al.* [104] introduced correction terms and extended the original model to a standard D2Q9 lattice. With the work of Zheng *et al.* [105]; however, the model was significantly improved because it was discovered that the free-energy approach could be used to recover the Cahn-Hilliard equation [106]

$$\frac{\partial\phi}{\partial t}+\boldsymbol{\nabla}\cdot(\phi\boldsymbol{u})=\boldsymbol{\nabla}\cdot\left(M\boldsymbol{\nabla}\mu_{\phi}\right). \tag{3.13}$$

The Cahn-Hilliard equation can be understood as an advection-diffusion equation for the order parameter used for the interface tracking, where M is the mobility of the interface and μ_ϕ describes the chemical potential. The big advantage of incorporating the Cahn-Hilliard equation is the implicit conservation of mass of the phase-field model. However, solving the Cahn-Hilliard equation also requires fourth-order spatial derivatives due to the chemical potential containing the laplacian of the order parameter. To provide the necessary order for the spatial derivative, information from an extended neighbourhood needs to be incorporated, which worsens the locality of the method [107, 108]. This poses a severe limitation, especially in large-scale systems.

3.3 The Conservative Allen-Cahn Model

Another fundamental contribution to multiphase models with the LBM was made by the work of He *et al.* [109, 110]. Their studies aimed to reformulate the LBM based on a pressure formulation rather than a density formulation. The reasoning behind this is that multiphase flows in the incompressible limits should not be targeted to fluctuations in density. Along with the adapted LBM, another set of PDF was used for the interface tracking between the fluid phases. The second set of PDFs can recover the Cahn-Hilliard equation. Thus, their model shares intrinsic similarities with the free energy formulation introduced in Section 3.2.3. While He *et al.* showed impressive results by simulating three dimensional Rayleigh–Taylor instabilities, limitations also held back the model. Most prominently, their model was limited to rather low-density ratios of $O(10)$, and a lack of mass conservation was observed. To overcome this limitation, Lee and Lin [111] modified the original model by introducing directional derivatives.

At the same time, another idea was to incorporate the Allen-Cahn equation for interface tracking directly. The Allen-Chan equation, however, was harder to incorporate due to a lack of mass conservation. Finally, this problem was solved by the work of Chiu and Lin [112] with a reformulation of the Allen-Cahn equation in conservative form,

$$\frac{\partial \phi}{\partial t} + \nabla \cdot \left(\phi \boldsymbol{u}\right) = \nabla \cdot M \left(\nabla \phi - \frac{1 - \left(\phi - \phi_0\right)}{W} \boldsymbol{n} \right). \tag{3.14}$$

which is called the conservative Allen-Cahn equation (CACE) with $\boldsymbol{n} = {}^{\nabla\phi}/_{|\nabla\phi|}$ being the unit vector normal to the liquid-gas interface. The major advantage of the CACE is that it only requires a second-order spatial derivative. In combination with the LBM the CACE was first solved in the work of Geier *et al.* [49]. Their work compared a formulation based on the single-relaxation-time (SRT) collision model with a formulation based on central moments. With the central moment formulation, it was possible to directly calculate the interface curvature $\boldsymbol{n}$ from the cell's local moments. While this improved the locality of the algorithm, it was later shown to jeopardise the accuracy of the model [113] On the other hand, Fakhari *et al.* [50] used an SRT formulation to solve the CACE with isotropic finite differences [114] to compute the curvature of the phase field. This idea showed great success in a large variety of academic and industrial cases with binary multiphase problems consisting of fluids with high density ratios (O(1000)) and under high Reynolds numbers [50, 54, 2, 115–118, 4]. The versatility of the model in both physical and performance aspects motivated this thesis to take a closer look. In the following section, we will show it in detail.

3.3.1 Phase Field Step

In Equation (3.14) phase indicator, ϕ, is used to keep track of the evolution of the interface between the binary fluid mixture. In this work, the fluid with the lower density ρ_L is indicated by $\phi_L = 0$, while the fluid with higher density ρ_H is indicated by $\phi_H = 1$. Thus, the interface location is defined by $\phi_0 = 0.5\left(\phi_L + \phi_H\right)$. In the literature, different values for ϕ can be used. For example, it was pointed out that a range between -0.5 and $0,5$ leads to better symmetry in the case of low-density ratios between the fluids [50]. However, choosing $\phi_L = 0$ eliminates error from the weakly compressibility of the LBM, which is especially important for the light phase in the case of high-density ratios. At equilibrium, the phase field profile in the direction normal to an interface located at x_0 varies as per a

hyperbolic tangent

$$\phi(\boldsymbol{x}) = \phi_0 \pm \frac{\phi_H - \phi_L}{2} \tanh\left(\frac{\boldsymbol{x} - x_0}{W/2}\right). \tag{3.15}$$

The discretisation with the LBM is done in a familiar way

$$\boldsymbol{q}_\mathrm{h} = \mathscr{T}(\boldsymbol{h}) \tag{3.16}$$

$$\boldsymbol{h}^* = \mathscr{T}^{-1}\left(\boldsymbol{q}_\mathrm{h} + \boldsymbol{S}^\phi\left(\boldsymbol{q}_\mathrm{h}^\mathrm{eq} - \boldsymbol{q}_\mathrm{h}\right) + \boldsymbol{q}_\mathrm{h}^\mathrm{F}\right). \tag{3.17}$$

However, with the crucial difference that $\boldsymbol{q}_\mathrm{h}^\mathrm{F}$ represents a source term leading to the CACE from Equation (3.14). It can be stated in the form

$$\boldsymbol{F}_\mathrm{h}(\boldsymbol{x}, t) = \frac{4\phi(1-\phi)}{W} \cdot \boldsymbol{n}, \tag{3.18}$$

which can be transformed to the collision space as shown in Section 2.2.4. This thesis uses the transformation based on the work of Guo *et al.* [80] unless stated otherwise. Through the usage of the source term, the zeroth order moment changes to recover the phase indicator

$$\phi(\boldsymbol{x}, t) = \sum_q h_q(\boldsymbol{x}, t). \tag{3.19}$$

Thus, the equilibrium of the phase field lattice Boltzmann (LB) step does not depend on the density as usual but on the phase indicator. Furthermore, the first-order moments are no longer conserved. This means the velocity can not be computed from this adapted LB step, but the velocity is rather an input parameter coming from a coupling step. Through the mobility of the CACE the relaxation time of the phase field LB step is calculated as

$$\tau_\phi = \frac{M}{c_s^2}. \tag{3.20}$$

After recovering the phase indicator, the density of the whole system can be calculated with a simple interpolation

$$\rho(\boldsymbol{x}, t) = \rho(\phi) = \rho_\mathrm{L} + (\phi(\boldsymbol{x}, t) - \phi_\mathrm{L})(\rho_\mathrm{H} - \rho_\mathrm{L}). \tag{3.21}$$

3.3.2 Hydrodynamic Step

An adapted pressure formulation of the LBM is used to recover the evolution of the pressure and velocity field, which can be stated in a familiar form

$$\boldsymbol{q}_\mathrm{g} = \mathscr{T}(\boldsymbol{g}) \tag{3.22}$$

$$\boldsymbol{g}^* = \mathscr{T}^{-1}\left(\boldsymbol{q}_\mathrm{g} + \boldsymbol{S}\left(\boldsymbol{q}_\mathrm{g}^\mathrm{eq} - \boldsymbol{q}_\mathrm{g}\right) + \boldsymbol{q}_\mathrm{g}^\mathrm{F}\right). \tag{3.23}$$

However, the difference comes in by an adaption of the equilibrium,

$$f_\mathrm{g}^\mathrm{eq}\left(p^*, \boldsymbol{u}, \boldsymbol{\xi}\right) = f^\mathrm{eq}\left(p^* - 1, \boldsymbol{0}, \boldsymbol{\xi}\right) + f^\mathrm{eq}(1, \boldsymbol{u}, \boldsymbol{\xi}). \tag{3.24}$$

This continuous formulation of the equilibrium needs to be integrated with the respective generating function as shown in Section 2.2.2. Since the density of the fluids is recovered by the phase indicator, it is no longer calculated from the zeroth order moment of the hydrodynamic LB step. The zeroth order moment herein is rather replaced by the normalised pressure $p^* = p/\rho c_s^2$, and reads

$$p^*(\boldsymbol{x}, t) = \sum_i g_i(\boldsymbol{x}, t). \tag{3.25}$$

To recover the incompressible NSE for multiphase flows, additional forcing terms must be applied during the collision of the LBM. These forcing terms

$$\boldsymbol{F}(\boldsymbol{x},t) = \boldsymbol{F}_s + \boldsymbol{F}_p + \boldsymbol{F}_\mu + \boldsymbol{F}_b, \tag{3.26}$$

are the surface tension force $\boldsymbol{F}_s$, the pressure force $\boldsymbol{F}_p$ and viscous force $\boldsymbol{F}_\mu$

$$\boldsymbol{F}_s(\boldsymbol{x},t) = \mu_\phi \boldsymbol{\nabla}\phi \tag{3.27}$$

$$\boldsymbol{F}_p(\boldsymbol{x},t) = -p^* c_s^2(\rho_\mathrm{H} - \rho_\mathrm{L})\boldsymbol{\nabla}\phi, \tag{3.28}$$

$$\boldsymbol{F}_\mu(\boldsymbol{x},t) = \nu(\rho_\mathrm{H} - \rho_\mathrm{L})\left[\boldsymbol{\nabla}\boldsymbol{u} + (\boldsymbol{\nabla}\boldsymbol{u})^T\right]\cdot\boldsymbol{\nabla}\rho. \tag{3.29}$$

The surface tension force is calculated with the chemical potential for binary fluids

$$\mu_\phi = 4\beta\frac{\phi(\phi-1)(2\phi-1)}{2} - \kappa\Delta\phi, \tag{3.30}$$

where the coefficients β and κ are given as

$$\beta = \frac{12\sigma}{W}, \qquad \kappa = \frac{3\sigma W}{2}, \tag{3.31}$$

with σ being the surface tension. An important feature of the LBM is that the deviatoric stress tensor can be computed using the non-equilibrium part of the second-order moments. In this way, it is possible to compute the viscous force cell locally as,

$$\boldsymbol{F}_\mu(\boldsymbol{x},t) = -\frac{\nu}{c_s^2\Delta t}\left[\sum_\beta c_{\beta i}c_{\beta j}\times\sum_\alpha \mathcal{T}_{\beta\alpha}\left(g_\alpha - g_\alpha^{\mathrm{eq}}\right)\right]\frac{\partial\rho}{\partial x_j}. \tag{3.32}$$

To further improve the locality of the model, no direct derivates of the density are computed, but they are rather approximated using the phase field

$$\boldsymbol{\nabla}\rho = \left(\rho_H - \rho_L\right)\boldsymbol{\nabla}\phi, \tag{3.33}$$

where the spatial derivatives of the phase field are computed employing second-order isotropic finite differences:

$$\nabla\phi = \frac{c}{c_s^2(\Delta x)^2}\sum_i \boldsymbol{c}_i w_i \phi(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t), \tag{3.34}$$

$$\nabla^2\phi = \frac{2c^2}{c_s^2(\Delta x)^2}\sum_i w_i\left[\phi(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t) - \phi(\boldsymbol{x},t)\right], \tag{3.35}$$

with $c = {}^{\Delta x}/_{\Delta t}$. The kinematic viscosity ν is computed with,

$$\nu = c_s^2\left(\tau\frac{\Delta t}{2}\right), \tag{3.36}$$

and the relaxation time τ is obtained from a linear interpolation with the phase indicator. It takes the form of

$$\tau(\boldsymbol{x},t) = \tau(\phi) = \tau_\mathrm{L} + (\phi(\boldsymbol{x},t) - \phi_\mathrm{L})(\tau_\mathrm{H} - \tau_\mathrm{L}). \tag{3.37}$$

where $\tau_\mathrm{H} \equiv \tau_\mathrm{H}(\boldsymbol{x},t)$ is the relaxation time of the heavy phase and $\tau_\mathrm{L} \equiv \tau_\mathrm{L}(\boldsymbol{x},t)$ is the relaxation time of the light phase. Notably, for the interpolation of the kinematic viscosity, different choices can be made; however, as pointed out by Fakhari *et al.* [50] Equation (3.37) gives a reasonable balance between accuracy and stability.

3.4 Three-Phase Contact Angle

Modelling wettability effects is one of the key points for multiphase simulations. This is because fluid-solid interactions occur practically in every real-world application, including flow through porous media, colloidal suspensions, or generally flow around solid structures [119]. As shown in Section 3.2.3, sometimes the wettability effect is even encoded directly in the definition of a multiphase model. In the course of this work, however, a contact angle is imposed rather in the style of an additional boundary condition acting on the phase indicator. For the contact angle between the fluid phase and the solid wall, the following definition is used [119],

$$\hat{\boldsymbol{n}} \cdot \nabla \phi|_{\boldsymbol{x}_w} = -\sqrt{\frac{2\beta}{\kappa}} \phi_w \left(1 - \phi_w\right) \cos\left(\theta\right), \tag{3.38}$$

where ϕ_w is the unknown value of the phase field in the wall node that needs to be determined to impose the desired contact behaviour. The outward-facing normal of the solid wall $\hat{\boldsymbol{n}}$ depends on the geometry of the solid boundary. Typically, in the case of non-moving geometries, it is reasonable to pre-calculate the normal direction during an initialisation phase [120]. The static contact angle, θ, must be an input parameter to impose a desired behaviour. For the unknown value of the phase field, ϕ_w, the following relations are applied,

$$\hat{\boldsymbol{n}} \cdot \nabla \phi|_{\boldsymbol{x}_w} = \left.\frac{\partial \phi}{\partial n_w}\right|_{\boldsymbol{x}_w} = \frac{\phi_f - \phi_s}{2h} \tag{3.39a}$$

$$\phi_w = \frac{\phi_f + \phi_s}{2}. \tag{3.39b}$$

With the geometrical normal direction $\hat{\boldsymbol{n}}$ the position of the corresponding fluid node can be defined as $\boldsymbol{x}_f = \boldsymbol{x}_s + \hat{\boldsymbol{n}}$. Furthermore, the value of the phase field in the fluid node is noted as $\phi_f = \phi(\boldsymbol{x}_f)$, and at the solid node as $\phi_s = \phi(\boldsymbol{x}_s)$. In addition to this, the distance between the boundary cell and the fluid cell is $2h = \left|\boldsymbol{x}_f - \boldsymbol{x}_s\right| = |\hat{\boldsymbol{n}}|$. By substituting Equation (3.39a) into Equation (3.38), the unknown value for the phase field at the solid node ϕ_s is,

$$\phi_s = \frac{1}{a}\left(1 + a - \sqrt{(1+a)^2 - 4a\phi_f}\right) - \phi_f, \tag{3.40}$$

where

$$a = -h\sqrt{\frac{2\beta}{\kappa}} \cos\theta. \tag{3.41}$$

Note that the contact angle $\theta = 90°$ is the trivial case in which $\phi_s = \phi_f$. This case is not defined by Equation (3.40) [2].

3.5 Extension to Thermocapillary Flows

Extending the binary fluid model to include temperature-driven motion allows for observing bubbles and droplets under the effect of so-called thermocapillary or Marangoni convection. Thermocapillary motion is crucial in many natural processes and various industrially relevant applications [121–124].

Due to the capability of the LBM to simulate multiphase systems, numerous scientists have extended it to capture thermocapillary effects [122, 124–128]. The extension of the CACE for capturing thermocapillary effects is done in two steps. The first step is to extend the surface tension force

$$\boldsymbol{F}_s(\boldsymbol{x},t) = \mu_\phi \nabla\phi(\boldsymbol{x},t) + \frac{3}{2}\xi\sigma_T\left(\left|\nabla\phi^2\right|\nabla T(\boldsymbol{x},t) - \left(\nabla T(\boldsymbol{x},t)\cdot\nabla\phi\right)\nabla\phi\right). \tag{3.42}$$

Contrary to Equation (3.27), the influence of a temperature field T is now added. This also adds a second spatial derivative that needs to be solved for the temperature field. In the case of this thesis, Equation (3.34) is employed similarly to stay consistent with the spatial derivative of the phase indicator. To impose an effect on the surface tension σ_T, a linear interpolation in the form of

$$\sigma(T) = \sigma_{\text{ref}} + \sigma_{\text{T}}\left(T - T_{\text{ref}}\right), \tag{3.43}$$

is used, where σ_{ref} is the surface tension at the reference temperature, T_{ref}, and $\sigma_T = \frac{\partial\sigma}{\partial T}$ is the rate of change of the surface tension with respect to the temperature, T [128].

The second step includes the evolution of the temperature field T within the simulation. This is done by solving the following heat equation

$$\frac{\partial T}{\partial t} = -\boldsymbol{u}\cdot\nabla T + \frac{1}{\rho c_{\text{p}}}\left(\nabla\kappa\cdot\nabla T + \kappa\nabla^2 T\right) + q_{\text{T}}. \tag{3.44}$$

The formulation employed here leads to reasonable results when heat dissipation and compression work done by the pressure are negligible [5]. The temperature equation is defined with the thermal diffusivity κ and the heat capacity c_{p}. For the multiphase system, these can be recovered with the help of the phase indicator,

$$\kappa = \kappa_{\text{H}} + \phi\left(\kappa_{\text{H}} - \kappa_{\text{L}}\right) \tag{3.45}$$

$$c_{\text{p}} = c_{\text{p,H}} + \phi\left(c_{\text{p,H}} - c_{\text{p,L}}\right). \tag{3.46}$$

Furthermore, a heat source density can be introduced with q_{T}.

Within the framework of thermocapillary LBM, there are numerous approaches to introduce the thermal field and its evolution. These range from coupling to a finite-difference-based Runge-Kutta integration scheme [128] to employing a thermal LBM on a lattice stencil [127]. These methods show only minor differences in the accuracy and computational efficiencies. In the scope of this thesis, we decided to use the LB based solver. More details on a comparison between the different possibilities can be found [5].

The LBM based solver for the temperature equation can be formulated as

$$\boldsymbol{q}_{\text{f}} = \mathscr{T}\left(\boldsymbol{f}\right) \tag{3.47}$$

$$\boldsymbol{f}^* = \mathscr{T}^{-1}\left(\boldsymbol{q}_{\text{f}} + \boldsymbol{S}^f\left(\boldsymbol{q}_{\text{f}}^{\text{eq}} - \boldsymbol{q}_{\text{f}}\right)\right) + \Delta t\, w_i\, \boldsymbol{q}_{\text{f}}^{\text{T}}, \tag{3.48}$$

where $\boldsymbol{f}$ denotes the thermal PDFs. The formulation of the LBM discretisation is similar to Equation (3.17) with the difference that the order parameter for the thermal solver is not the phase-field, ϕ, but the temperature, T. Thus, the temperature field can be recovered using the zeroth-order moment of the thermal PDFs,

$$T(\boldsymbol{x},t) = \sum_i f_i(\boldsymbol{x},t). \tag{3.49}$$

To recover the CACE, it was necessary to introduce a source term according to Equation (2.36). For the thermal LBM formulation, a source term is only necessary if a heat source, q_T, exists. This can be, for example, a laser source as it will be examined in Chapter 8. The relaxation rate for the thermal LBM can be calculated as,

$$\tau_T = \frac{1}{\frac{1}{2} + c_s^2 \cdot \kappa}, \tag{3.50}$$

and is related to thermal conductivity.

3.6 Complete Algorithm

The complete algorithm to simulate multiphase flows using the conservative Allen-Cahn model (CACM) is given through Algorithm 2. The algorithm starts by initialising the phase-field and hydrodynamic PDFs. As a next step, the domain and obstacle boundaries are mapped to a flag field. With this, the main loop over the domain cells begins. In this loop, the first step is communicating the hydrodynamic PDFs $\boldsymbol{g}$ using non-blocking message passing interface (MPI) routines. While the data communication is performed, the LB algorithm for the phase-field PDFs $\boldsymbol{h}$ can be executed (compare Section 2.3. Afterwards, the phase-field PDFs $\boldsymbol{h}$ are communicated, while the LB algorithm for the hydrodynamic PDFs $\boldsymbol{g}$ is performed.

During the update of the phase-field PDFs, $\boldsymbol{h}$ their zeroth-moment is directly evaluated and written to a temporary phase-field array. A temporary array is necessary to avoid a second pass over the PDFs. A pointer swap updates the phase-field array after the hydrodynamic LB step. With this, the velocity field $\boldsymbol{u}$, updated during the hydrodynamic LB step, and the phase-field ϕ are updated synchronously. The algorithm ends by calculating the three-phase contact angle for the phase field and synchronising the phase field with neighbouring subdomains. This is necessary because the phase field is accessed with a finite difference stencil in both LB-steps.

Algorithm 2: Complete algorithm for the CACM.

```
1  Initialise PDFs h, g (Section 2.3)
2  Initialise flag field with bounding walls and obstacles (Section 4.4.5)
3  for each time step t do
4      Start synchronise hydrodynamic PDFs g (Section 4.4.4)
5      Apply boundary conditions for phase-field PDFs h (Section 2.2.3)
6      Perform stream-collide in each cell for phase-field PDFs h (Equation (3.17))
7      Wait for synchronise to finish
8      Start synchronise phase-field PDFs h (Section 4.4.4)
9      Apply boundary conditions for hydrodynamic PDFs g (Section 2.2.3)
10     Perform stream-collide in each cell for hydrodynamic PDFs g (Equation (3.23))
11     Swap pointers of phase-field φ source and destination array
12     Apply contact angle for phase-field φ (Section 3.4)
13     Wait for synchronise to finish
14     Synchronise phase-field φ (Section 4.4.4)
15 end
```

The extension of Algorithm 2 for the simulation of thermocapillary flow problems is given through Algorithm 3. Both algorithms show a similar structure. The main difference is that a third LB step is executed in Algorithm 3 to solve the energy equation. It is important to note that the third LB step does not rely on the phase-field ϕ. Hence, it is possible to overlap the synchronisation of the phase field with updating the thermal PDFs. In this way, all MPI communications can be hidden behind computations.

Algorithm 3: Thermocapillary algorithm using an LBM solver for the heat equation.

```
1  Initialise PDFs h, g and f (Section 2.3)
2  Initialise flag field with bounding walls and obstacles (Section 4.4.5)
3  for each time step t do
4      Start synchronise thermal PDFs f and temperature T (Section 4.4.4)
5      Apply boundary conditions for phase-field PDFs h (Section 2.2.3)
6      Perform stream-collide in each cell for phase-field PDFs h (Equation (3.17))
7      Wait for synchronise to finish
8      Start synchronise phase-field PDFs h (Section 4.4.4)
9      Apply boundary conditions for hydrodynamic PDFs g (Section 2.2.3)
10     Perform stream-collide in each cell for hydrodynamic PDFs g (Equation (3.23))
11     Swap pointers of phase-field φ source and destination array
12     Apply contact angle for phase-field φ (Section 3.4)
13     Wait for synchronise to finish
14     Start synchronise hydrodynamic PDFs g and phase-field φ (Section 4.4.4)
15     Apply boundary conditions for thermal PDFs f (Section 2.2.3)
16     Perform stream-collide in each cell for thermal PDFs f (Equation (3.48))
17     Wait for synchronise to finish
18 end
```

CHAPTER 4

SOFTWARE STACK

Contents

This chapter gives an overview of the software stack used and developed in this work. After motivating the code generation approach in Section 4.1 employed here, the Python package PYSTENCILS *is introduced in detail in Section 4.2. Building on the work of Bauer et al. [28]* PYSTENCILS *has been extended with an enhanced typing system to allow for mixed-precision computations and support for AMD general purpose graphics processing units. Following the introduction of* PYSTENCILS, LBMPY *is shown in Section 4.3 as the building block that describes and derives state-of-the-art variants of the lattice Boltzmann method. In the scope of this thesis,* LBMPY *has been redesigned completely. This concerns the derivation of all collision models by introducing more modularity and allows the expression of most complex models, like the K17 cumulant collision model with a minimum amount of floating point operations. Furthermore, all common streaming algorithms are now fully supported, and extended boundary conditions have been implemented to account for complex geometries by employing interpolation routines. Finally, phase-field models based on the conservative Allen-Cahn equation have been integrated. To realise large-scale simulations, an integration of the code generation pipeline into the* WALBERLA *framework is realised. This integration is explained in detail in Section 4.4.5 after introducing* WALBERLA *in Section 4.4 and its most important modules. In the scope of this thesis, the interface between* PYSTENCILS *and* WALBERLA *has been extended to support complex simulation setups with a refined grid structure on general purpose graphics processing units.*

4.1 Code Generation

In the field of computer science, every tool that generates code in some programming language can be referred to as a code generator. Prominent examples of such tools would be compilers, e.g., the GNU Compiler Collection (GCC) [1] to generate assembly code from a higher level language like C++. Over the past eight centuries, it has been proven highly useful to express a problem in a higher abstraction and then use some software to make it accessible to the computing architecture [129]. However, making the problem not only accessible to a certain computing architecture but also producing fully optimised machine code is an inherently complex task. The complexity of this task strongly increases depending on the generality of these abstractions.

In the category of highly generalised abstractions fall, for example, general-purpose languages like C++ that give developers great flexibility by allowing them to manipulate memory manually if they desire to do so. On the other side, complex abstractive constructs can be built using the full toolset offered by the object orientation of the language. If not done carefully, such constructs might be impossible for a general-purpose compiler to optimise, thus severely impacting the program's performance. Therefore, applying optimisations at compile time is often necessary to eliminate their impact at runtime. In C++, however, other than by modifying the compiler, this is only possible in a limited way, e.g., by using static polymorphism via templates.

Furthermore, increasing the generality of problems also results in an extended set of parameters that will be provided at runtime rather than compile time. While this increases the usability, it may impact the program performance negatively because it might be harder for the compiler to remove branches or simplify terms. Eventually, this may lead to a reduction in the prefetching and pipelining. Parameters must be provided as constants in code files instead of comprehensively structured, custom configuration files to enable compile-time optimisations.

[1] https://gcc.gnu.org/

An alternative to solving these drawbacks is using a domain-specific language (DSL) with a code generator. A DSL is a custom language tailored to express specific problems in a formulation close to the problem itself. In this way, the DSL does not cover programming details, allowing the user to work more abstractly than using a general-purpose language. Since the DSL is limited to specific problems, it is possible to apply domain-specific optimisations and thus generate nearly optimal code for problems covered by the DSL. While it is also possible to directly generate machine code from a DSL, we will generate code for another language and then utilise existing compilers to get to the machine code [130]. This approach has the advantage that the output of the code generator is still readable and that we can build upon the facilities of existing compilers to keep our generation tool simple. Thus, the DSL serves as a platform to encapsulate the intricacies of our problem domain in a concise and manageable way. Through the DSL, we can succinctly describe the nuances of our scenario, facilitating rapid prototyping and experimentation. This approach abstracts the complexities of specialised optimisations, allowing us to focus on conceptualising the problem rather than its implementation details.

4.2 PySTENCILS

The evolution of a physical system is often modelled using partial differential equations (PDEs). Since the analytical solution of PDEs is rarely accessible for relevant real-world problems, numerical tools must be used to approximate them. Some of the most important tools follow the approach of discretising the domain on which the problem is defined into small pieces of space and time, and then the target equation is solved on each of the pieces. These discretisation techniques can be broadly characterised into two classes: the problem is discretised on a finite unstructured or structured mesh. A significant advantage of unstructured approaches is that they fit the mesh well to complex geometries. Thus, numerical errors at the boundary of the geometry can be minimised. However, this flexibility also leads to the problem that the meshing process is a complex task and that updating the mesh in a timestep process might be more costly because the geometrical information needs to be loaded from memory for each element. Hence, especially for large-scale problems, a structured grid is often beneficial. In structured grid approaches, the underlying data can be updated using identical operations for each array point. Using regular nested loops to cover the domain, each array point can be accessed directly and consecutively.

This is the fundamental idea of the PySTENCILS code generator. Employing the Python programming language for our embedded DSL, a model developer can focus on the set of operations that will be applied on each point. This is done in symbolic form using the computer algebra system *SymPy* [131]. Subsequently, we transform this high-level formulation into a lower-level representation closer to the target architecture. These transformations bridge the gap between the abstract problem description and its eventual implementation. By systematically converting the DSL-based formulation into a problem-specific, highly optimised piece of code, we ensure that the complexity of specialised optimisations is handled efficiently and effectively.

Adopting this strategy, we streamline the implementation process and ensure that our codebase remains maintainable over time. This is particularly crucial when dealing with large-scale simulations and intricate problem domains where specialised optimisations can quickly become unwieldy. Thus, by integrating the DSL approach, we balance complexity and maintainability, ultimately enabling us to tackle challenging computational problems confidently. In Section 4.2.2, details of the code generation process will be shown.

4.2.1 Related Work

Before going into details of the software structure of PYSTENCILS, we show some packages that share similar ideas to PYSTENCILS. This section is intended to review them and provide an overview of the state of the art. Please note that it is by no means complete, as it only focuses on the most relevant projects and publications.

Loo.py

Loo.py is a programming system based on Python. It defines a data model for array-style computations and a library of transformation operating on this model [132]. Optimisations such as loop tiling, vectorisation, storage management, and unrolling or changing data layout are supported. To understand the core idea of Loo.py, a small code snippet is provided in listing 4.1. This snippet shows the main components of creating a Loo.py kernel. The first component, called *loop domain*, is formulated with the help of the Integer Set Library (isl) [133]. The second component is formed by the *instructions*. These are scalar assignments with left and right-hand sides inside a *loop domain*. From this description, Loo.py generates highly efficient code for central processing unit (CPU) and general purpose graphics processing unit (GPGPU) platforms. The kernels can be executed directly within Loo.py using PyOpenCL [134].

CODE LISTING 4.1: Illustration of the main components of a Loo.py kernel. Code snippet taken from [132].

```
knl = loopy.make_kernel(
      "{ [i]: 0<=i<n }", # loop domain
      "out[i] = 2*a[i]") # instructions
```

ExaStencils

The code generation framework ExaStencils uses whole program generation with a layer design pattern to generate highly optimised and massively parallel geometric multigrid solvers on (block-)structured grids [135]. The framework is written using the Scala programming language. Multiple DSL layers allow users to define problems highly abstract using a LaTeX-like syntax on the top layer and give more and more precise control on further levels. The resulting program can be generated with message passing interface (MPI), OpenMP or CUDA to support NVIDIA GPGPUs. With the GHODDESS (Generation of Higher-Order discretisations Deployed as ExaSlang Specifications) framework and extension to ExaStencils exist [136]. It allows the use of *SymPy* for describing problems with the discontinuous Galerkin method.

SymForce

The SymForce package builds on *SymPy* to represent non-linear optimisation problems for robotics applications [137]. The development speed and flexibility of the mathematical environment are combined with highly optimised implementations in C++ using a code generation approach. The key benefits of the framework are rapid prototyping, code flattenings across function calls and automatic differentiation. Furthermore, the heavy use of common subexpression elimination (CSE) allows generating low-level code with a minimum of computational operations.

YASK

YASK (Yet Another Stencil Kit) is a software framework developed by Intel [138]. It provides a DSL to describe stencil problems and comes with a compiler to create C++ source code. The compiler applies highly advanced code optimisations such as multilevel loop interchanges, vector-folding or spatial- and temporal-blocking. Through OpenMP offloading, GPGPUs are supported directly, and MPI allows the usage of multiple processes for solving large problem sizes.

4.2.2 Abstraction Layers

The software design of PyStencils employs multiple layers of abstractions, illustrated in Figure 4.1. Building the framework with a layer concept enhances the modularity and creates a separation of concerns in the framework. Thus, users working on mathematical modelling do not need to know the intrinsics of DSL-specific optimisations. The top layer of PyStencils is formed by the PDE-layer. On this level, custom classes can describe PDEs in their continuous form. The continuous form is close to the mathematical description typically found in the literature. Discretising the continuous formulation of the PDE, e.g.,

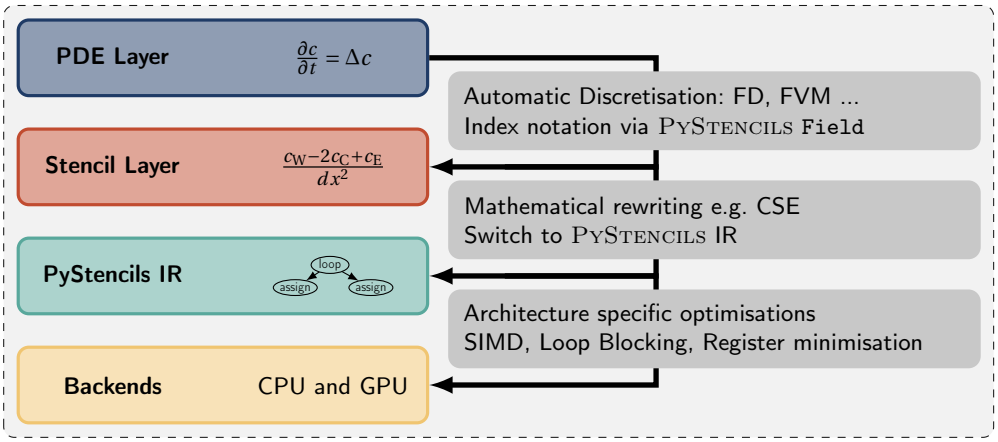


Figure 4.1: Abstraction layers of the PyStencils code generation package. On the top level, mathematical problems can be described in their continuous form as PDE. The second layer represents the discretised version of these, thus introducing the index notation to describe relative data accesses. From there, the symbolic representation gets transformed to the PyStencils intermediate representation (IR), which will be printed to low-level code.

by applying a finite differences scheme, leads to the stencil layer of PyStencils. The

stencil layer uses the `Field` class of PYSTENCILS to describe stencils in index notation. At this point, the abstraction layers of PYSTENCILS are already useful because complex stencils that the automatic discretisation facility might not support can be formulated manually. This abstraction layer still fully utilises *SymPy* as a computer algebra system. In fact, for each indexed variable, a different *SymPy* symbol is used. In this way, the full set of mathematical rewriting is still accessible.

As a next step, the symbolic description of the stencil layer is transformed to the PYSTENCILS intermediate representation (IR). This tree structure is much closer to lower-level C code by introducing loop structures, control flow or abstract syntax tree (AST) nodes for custom code injection. Furthermore, type information is added as needed by the statically typed C language. Architecture-specific optimisations like single instruction, multiple data (SIMD) vectorisation, loop blocking or register minimisation are executed at this level. After applying architecture-specific optimisations and thus reaching the final AST of the PYSTENCILS IR, it will be printed by a respective backend.

A standard example will be explored to highlight the details of the abstraction layers of PYSTENCILS and showcase the general workflow. To do so, an advection-diffusion equation (ADE) will be explored. The ADE defined as

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - \nabla \cdot (\boldsymbol{u} c), \tag{4.1}$$

is a transport equation that describes the temporal evolution of the order parameter c under the influence of advection and diffusion, where the diffusion process is controlled by the diffusivity D and the convection process by the velocity $\boldsymbol{u}$. The ADE is considered here due to its strong relation to the Navier-Stokes equation (NSE) and thus to its great importance in computational fluid dynamics (CFD). In fact, the NSE can be understood as an ADE for the fluid momentum $\rho \boldsymbol{u}$ [33]. The ADE has many important applications as the order parameter can describe temperature for heat problems (see Section 3.5) or a mass concentration, among many others. For example the spread of the SARS-CoV-2 virus has been analysed in a supervised bachelors thesis during this work [10], where the virus was modelled with Equation (4.1)

PDE Layer

The PDE layer of PYSTENCILS accepts PDEs in their continuous form. A code example of how this can be done is illustrated in listing 4.2. The ADE shown in Equation (4.1) consists of two array quantities c and $\boldsymbol{u}$, and one scalar value D. The scalar value D is directly defined by using a *SymPy* `Symbol`, but also numerical values could be stated here to fix the ADE to a specific diffusivity. For the array quantities, the `Field` class of PYSTENCILS is used. The `Field` class represents a high-level description of a multidimensional array. Thus, it contains information like the spatial dimensions, the shapes and strides in each dimension, a C data type or the memory layout. The memory layout can either be row-major or column-major order. Important to note here is that most information (like the shapes and strides) can be defined either numerically for a specific size or using symbolic variables. Another speciality is that the `Field` class contains a subclass that describes an array-access relative to loop indices x, y, z. The three building blocks to describe the ADE can be created with the defined variables. The transient term $\frac{\partial c}{\partial t}$ is represented by the `Transient` class, the diffusive term $\nabla \cdot (D \nabla c)$ by the `Diffusion` class

and the advective term $\nabla \cdot (\boldsymbol{u}\phi)$ by the `Advection` class. These classes inherit from the `Function` class of *SymPy*. Therefore, it is possible to define the final PDE completely as a symbolic expression as shown in listing 4.2, which allows the application of the full set of mathematical manipulations offered by the computer algebra system.

Code Listing 4.2: Stating the ADE in the PDE layer of PyStencils. The array quantities are declared using the `Field` class, while the diffusivity D is declared as *SymPy*-`Symbol`.

```
import pystencils as ps
import sympy as sp

dim = 2
c, u = ps.fields(f"c, u({dim}): [{dim}d]")
D = sp.Symbol("D")

pde = ps.fd.transient(c) - ps.fd.diffusion(c, D) + ps.fd.advection(c, u)
```

Stencil Layer

At this point, it is already possible to fix model-specific parameters. For example, the diffusivity D could be fixed to a problem-specific numerical value. On the other side, it is possible to add additional terms easily using the described building blocks [26].

Applying a suitable discretisation scheme to the PDE defined above leads to the next abstraction layer of PyStencils. At the stencil layer, an update rule is formed using index notation. This means a set of algebraic equations that contain an array of elements with their access information encoded. The access information is expressed by the `Field.Access` class, which is a subclass of `Field`. The stencil layer again makes heavy use of *SymPy*, offering a wide range of mathematical tools. Among others, these are mainly manipulations like expansion, factorisation, and simplification, but also powerful tools like the CSE, that can significantly reduce terms, which will eventually lead to a reduction of floating point operations (FLOPs) later. For the ADE defined earlier, a discretisation based on a finite difference approximation can be derived directly using high-level functions of PyStencils. The code snipped shown in listing 4.3 should illustrate how this can be done. As a first step, a suitable discretisation class is chosen. A second-order finite

Code Listing 4.3: Using the finite differencing facility of PyStencils to discretise Equation (4.1) automatically with a second-order accurate approach.

```
dx, dt = sp.symbols("dx dt")
discretize = ps.fd.Discretization2ndOrder(dx, dt)
discretization = discretize(pde)
```

differencing scheme is applied using the `Discretization2ndOrder` class. This class gets the spatial and temporal resolution as input parameters. Afterwards, the resulting object can be called using the defined PDE. This leads to the following output:

$$c_\mathrm{C} + dt\left(-\frac{c_\mathrm{E}u^0_\mathrm{E} + c_\mathrm{N}u^1_\mathrm{N} - c_\mathrm{S}u^1_\mathrm{S} - c_\mathrm{W}u^0_\mathrm{W}}{2dx} + \frac{D\,(-4c_\mathrm{C} + c_\mathrm{E} + c_\mathrm{N} + c_\mathrm{S} + c_\mathrm{W})}{dx^2}\right). \tag{4.2}$$

The spatial access is expressed with the subscript, while the superscript of the symbolic variables indicates the cell index. For example, u^1_N represents an access on the second component of u in the northern direction. The direction-on-compass notation shown here translates to a numerical index. `Fields` can be annotated with their desired memory

layout, i.e., linearisation order. However, arbitrary memory layouts, including the special cases of structure of arrays (SoA) and array of structures (AoS) layouts, are supported. If the memory layout is known, the code generator can arrange the loops iterating over the array such that the array accesses are consecutive in memory. The field's shape and strides can be specified at runtime via kernel parameters or at code generation time. Providing fixed shapes and strides can help the later compiler pass to produce more optimised and tailored machine code.

Expressions leading to branching in the kernel are intentionally not part of the stencil layer except for piecewise-defined functions. Piecewise-defined functions are an ordered list of condition-value pairs together with a mandatory fallback value if none of the conditions evaluates to true. They could either be represented by nested ternary `if` operators, but the main advantage of this formulation is that in SIMD vectorised code, these piecewise-defined functions can be efficiently mapped to masking and blending instructions. The following sections will give an overview of the SIMD vectorisation facility.

PYSTENCILS IR

After deriving the discretisation for the ADE, the next stage is to transform it to the IR of PYSTENCILS. An example code snipped to give the basic idea is shown in listing 4.4.

The transformation to the PYSTENCILS IR is executed by calling the `create_kernel` function. It receives a list of assignments in static single-assignment (SSA) form. An assignment is an AST node holding a left- and right-hand side. In the case of this example, the right-hand side is the symbolic expression of the discretized ADE, while the left-hand side is a symbolic field access for a temporary array that is used to store the updated values in successive timestep updates. The assignments are depicted within a syntax tree provided by the *SymPy* framework, wherein the leaf nodes comprise either symbols, numeric values or field accesses. In Figure 4.2, an extraction of the AST representing the diffusion part of the ADE is illustrated.

Several steps are needed to obtain the PYSTENCILS IR. The first step is to check the consistency of the input list of assignments. The most important conditions that are checked at this stage are the SSA form of the assignments and the independence of array writes. Enforcing SSA form means that every symbol can only appear once on the left-hand side of an assignment. Similarly, an array must only be written at one spatial location per field index. Additionally, a thread safety check indicates if the assignments can be parallelised using OpenMP or GPGPUs.

CODE LISTING 4.4: Creating a compute kernel from an assignment. As a first step, the assignment (which can also be a list of assignments) is transformed to an AST with the PYSTENCILS IR. This is done using a configuration class called `CreateKernelConfig`. It holds all the necessary information, like the target architecture or the instruction set the output should use. Once the AST is formed, it can be directly compiled within the Python environment.

```
cn = ps.fields(f"c_n: [{dim}d]")
assign = ps.Assignment(cn.center(), discretization)

simd = {'instruction_set': "avx512"}
config = ps.CreateKernelConfig(target=ps.Target.CPU, cpu_vectorize_info=simd)
ast = ps.create_kernel(assign, config=config)
kernel = ast.compile()
```

After the consistency check, data types according to the C programming language are introduced. By design, a PYSTENCILS `Field` must hold a data type, which is done by employing the typing system of *NumPy* [2]. Thus, the green nodes in Figure 4.2 have known data types at the start. Consequently, data type information is only missing for free symbols and numeric constants (marked in red and yellow). In these cases, the following strategy is applied. The AST is traversed recursively, and each leaf returns its type information if known. If the type information is not known, a default type is applied. The default type is globally defined with the configuration class passed to the `create_kernel` function. Once a node has the type information of all its children collected, type collation is performed. It works by choosing the most general data type in a sequence of data types. For example, a list of integer types containing a floating pointing type will collate with the floating point type. When a common data type for a node is found, all its child nodes are cast to the common data type. Casting works by utilising a special PYSTENCILS AST node, which receives an expression and a data type to which the expression should be cast to. Lastly, the left-hand side of each assignment is compared with the common data type of the right-hand side. If the data types differ, a data cast is introduced. Utilising data casts in this way ensures type correctness, while a later compiler stage removes casts and optimises the resulting data types.

Next, the separate tree representations of all assignments are incorporated into a full syntax tree. Assignments nodes are inserted as children of a `Block` node. A `BlockNode` is an ordered collection of child nodes and comprises a scope. It is used as the only descendant in more complex nodes like loops or conditionals. The C backend prints a `Block` as a code section surrounded by curly braces. At this stage, the code generator checks that all accessed fields have the same spatial shape. If the fields have fixed spatial shapes, the test is executed right away, while for variable-sized fields, runtime checks in the form of assertions are inserted in the resulting code. Since assertions generally can be deactivated by compilers, inserting them at suitable places in the code to ensure code correctness during testing with no overhead at production runs is good practice. Next, loop nodes are created around the assignments. Optimal loop order is determined depending on the memory layout of accessed fields, such that the innermost loop iterates over the coordinate stored consecutively in memory to use the hardware prefetching capabilities best. The spatial shapes of the fields and their field accesses make it possible to determine correct loop bounds automatically and to avoid out-of-bounds memory access. Thus, the code generator automatically determines the minimum amount of ghost (halo) layers required at each field border. Communication or boundary routines must fill these ghost layers every time the compute kernel is run. Again, the communication routines can be created automatically by analysing the field accesses inside the original list of assignments.

The only way to express conditional evaluations on the algebraic level is through piecewise-defined functions. They must have a base case allowing efficient SIMD vectorization with blending instructions. However, a custom node is provided to express conditionals at the AST level. This `Conditional` is most prominently used to check for out-of-bounds access in GPGPU kernels. The array index is commonly calculated from a pool of threads spawned on a virtual block grid in these cases. Each of the threads must be checked for their validity. In general, branching is usually avoided due to the performance impact it can cause when introduced in inner loop passes. Especially to apply boundary

[2] `https://numpy.org/`

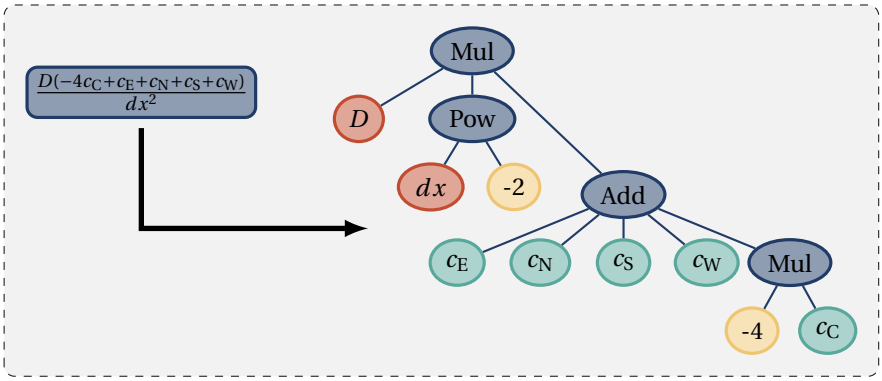


Figure 4.2: Example AST for the discretised diffusion part of Equation (4.1). The nodes of the AST are formed either by PYSTENCILS or *SymPy* classes, and the leafs are either symbols (red), numeric constants (yellow) or field access (green)

conditions PYSTENCILS offers the facility to create indexed kernels instead of a full domain kernel. An indexed kernel contains only a single loop over an index array, which stores coordinates and other information. In this way, the location of boundary cells can be stored inside the index vector, and their update can be separated from the domain update in a small and efficient kernel.

The last step to finish the AST in the PYSTENCILS IR is to resolve all field accesses. In the higher layers of the software, these have been expressed with the `Field.Access` class. At this stage, however, array accesses must be expressed by a linearised index, which depends on the loop counters and the array strides. Thus, at this point, a special AST node is introduced with a symbolic expression for the linearised index and a base symbol that will translate to a C pointer. In this way, it is possible to introduce temporary pointers at the outer loop levels to simplify the index calculation in the inner loop. In some cases, this can lead to performance increases. However, in the scenarios analysed in this thesis, the integer calculation of the linear index generally only plays a minor role; thus, in all benchmarks, no temporary pointers are introduced.

Backends

After building the AST in the PYSTENCILS IR, it is passed to one of the backends. The backends serve three purposes. The first purpose is to introduce architecture-specific optimisations, the second is to print the final kernel code to be compiled on a target architecture, and the last is to execute the resulting kernel.

In PYSTENCILS, different optimisations tailored to a specific hardware are implemented. For example, many stencil kernels can profit from spatial loop blocking. Applying loop blocking means subdividing the domain loops to keep certain variables longer in the register [139]. Herein, the optimal size of the blocking loop depends on the cache size of the target chip. Due to the code generation approach, different loop sizes can be tested quickly.

Another important optimisation is the usage of SIMD instructions. These are specialised vector instructions designed to increase the throughput of a chip by applying a certain instruction to a vector of data at once instead of a single datum. While general-purpose compilers usually introduce these instructions automatically (when the highest optimisation level is activated), they still fail to do so in complex cases. Thus, even in

recent literature, to ensure optimal results, SIMD instructions are introduced manually in the kernel code [140]. Within PyStencils, a simple yet powerful strategy is employed to directly introduce SIMD instructions in the resulting code. After creating the PyStencils IR, a second pass through the AST replaces scalar data types with vector types, AST nodes with their scalar nodes with their SIMD counterparts and field access with vectorised load and stores. The philosophy here is to not aim for general support of all possible SIMD instructions but only support a subset important for stencil codes and extend on demand. Like this, the SIMD instructions can be provided in a simple dictionary-like structure to map scalar instructions to vector instructions. At the beginning of this thesis, PyStencils only supported the x86 instruction set. At this point, PyStencils supports all relevant instruction sets, including SSE, AVX, AVX512, Neon, SVE, SME, RISC-V, and VSX. Thus, the latest Intel- AMD- and ARM-chips instruction sets are fully supported within PyStencils plus various less common ones.

Advantages of the AST structure of PyStencils were already shown in previous sections. For example, by analysing read and write accesses, minimal halo layers and optimal loop bounds can be determined. In Section 4.4, this information will also be used to automatically generate MPI packing and unpacking routines for the waLBerla framework (more details are provided in Section 4.4.4). Similarly, analysing the AST opens the possibility of applying automatic performance modelling. Recently, this path was explored for performance prediction on GPGPUs [6]. An overview of the idea is illustrated in Figure 4.3. The address expressions from the PyStencils AST are combined with hardware information and a suitable performance model. Finally, with this information, a performance prediction can be made before the kernel is executed. This workflow gives several major advantages. The most dominant advantages are the rapid analysis of different code generation configurations without executing the final compute kernel and additional insights to optimise performance bottlenecks. A detailed description of *Warpspeed* in combination with PyStencils can be found in reference [6].

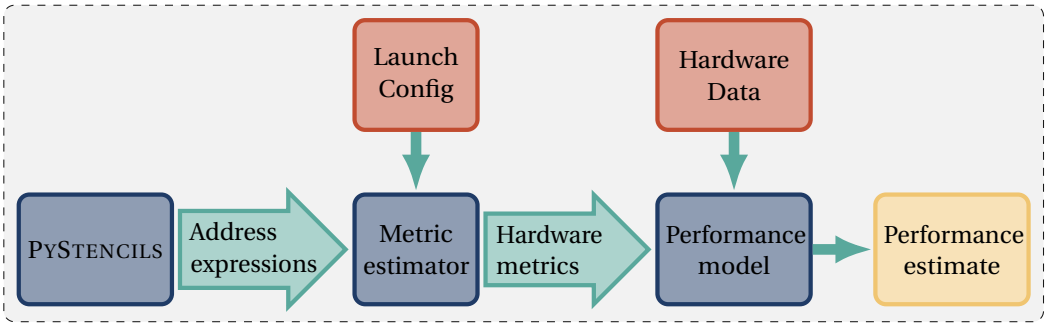


Figure 4.3: Workflow for automatic performance prediction using *Warpspeed* and PyStencils as code generator. The figure is inspired by reference [6]

After applying hardware-specific optimisations, a code printer of the backend prints the AST to a valid C code. At the time of writing, two code printers are supported. Both of the code printers are close to the C language. One specialises in CPU architectures, and the other in GPGPU architectures. Using only primitive C language features results in a highly versatile output that can be combined with many other languages or frameworks (see Section 4.4). The printed kernel code can be directly executed within the Python environment. The CPU backend uses system calls to a compiler specified within a con-

figuration file. Standard configuration files with standard compiler flags are provided for the most widespread platforms (Linux, Mac OS and Windows). The standalone library is created by embedding the printed AST in a template using the Python C application programming interface (API) [3]. The compiled library can be provided as a Python function, which receives *NumPy* arrays and numerical values.

Using GPGPUs is made possible by employing *CuPy* [4]. *CuPy* comes with its own compiler detection and supports NVIDIA and AMD GPGPUs. The major advantage of using *CuPy* here is that the package handles low-level device calls, for example, to transfer data from host to device. This allows to manipulate memory on a very high level in Python. Using *NumPy* and *CuPy* in conjunction with the generated compute kernels from the PYSTENCILS AST results in an incredibly powerful and versatile framework because stencil expressions can be realised within highly optimised compute kernels provided as Python function, while general mathematical manipulations can be realised using *NumPy* or *CuPy* directly on the data. Thus, even specific operations or evaluations can be done within a high level of abstraction.

4.3 LBMPY

As shown in Section 2.2, a wide variety of different lattice Boltzmann methods (LBMs) have been established in the last few decades. Oftentimes, they follow the same principle but differ in adding additional steps like transforming the populations to a different collision space to increase the stability and accuracy of the method. Furthermore, many of these methods have reached a level of complexity that a derivation by hand has become unfeasible. Therefore, the LBMPY package provides a generic, purely symbolic development environment for LBM. At its heart, the LBM employs a stencil structure; thus, it is an ideal candidate for automatic code generation using the PYSTENCILS package. In fact, the combination with PYSTENCILS makes LBMPY an incredibly versatile and powerful tool. Similar to the finite difference example shown in Section 4.2, LBMPY derives the LBM stencil update rules for PYSTENCILS.

4.3.1 Abstraction Layers

Much like the PYSTENCILS package, LBMPY adopts a layered approach as a key design principle. However, unlike PYSTENCILS, the layered structure of LBMPY emerges inherently from the LBM. An overview of the various software layers is provided in Figure 4.4. The first layer involves the definition of the LBM. Depending on the chosen collision space, basis polynomials, and lattice stencil, an appropriate discretisation of Equation (2.5) is dynamically derived. Additional options, such as specifying the relaxation rates, allow further refinement or extension of the resulting equations.

As a next step, a full collision operator is derived. If needed, the collision operator adds transformations to other collision spaces and mathematical simplifications, e.g., to eliminate certain terms. At this layer, additional adaptions of the collision rule are introduced. An example of such an adaption would be an large eddy simulation (LES) model that takes the original equations and inserts additional expressions to update the second-order relaxation rate locally depending on the LES model.

[3] https://docs.python.org/3/c-api/index.html
[4] https://cupy.dev/

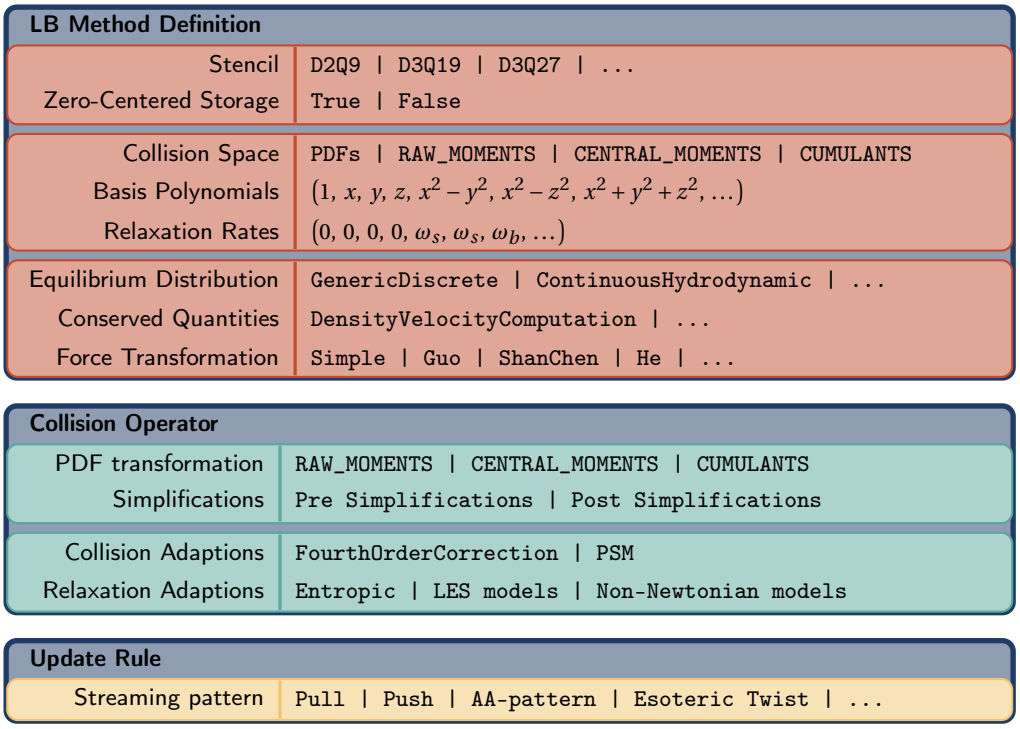


Figure 4.4: The different software layers of the LBMPY package. At the framework's core is the definition of the LBM. This is done by deriving a suitable discretisation of Equation (2.5) using the base polynomials that the user can choose. After deriving the basic discretisation, adaptions and optimisations can be applied to form a final collision rule. Adding a suitable streaming scheme will lead to the final update rule passed to the PYSTENCILS code generator.

Until this point, only the cell-local collision step is defined. This means that LBMPY functions entirely independently of PYSTENCILS to this stage. However, the streaming step needs to be inserted as the next step to finalise the ingredients of the LBM. The streaming step adds non-local information that can be introduced best using index notation via the PYSTENCILS `Field` class. A detailed description of the individual layers will be given in the following sections.

4.3.2 Method Definition

The core layer of LBMPY is an automatic procedure that takes an abstract LBM specification and derives a sequence of symbolic equations implementing Equation (2.6). This specification is formulated using a flexible Python API, which again makes heavy use of the computer algebra system *SymPy* [131] to represent and manipulate all components of an LBM in symbolic, mathematical form. The automatic derivation closely follows the theory outlined in Section 2.2.

A part of the parameter space of LBMPY's abstract method specification is shown in the red boxes in Figure 4.4. The lattice structure is defined by selecting a stencil and a storage format (absolute or zero-centered). In this context, the storage format refers to how the particle distribution functions (PDFs) are stored. It comes from the realisation

that Equation (2.5) can be decomposed

$$f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \boldsymbol{\xi}\right) = f^{\mathrm{eq}}\left(\rho_0, \boldsymbol{u}, \boldsymbol{\xi}\right) + f^{\mathrm{eq}}\left(\delta\rho, \mathbf{0}, \boldsymbol{\xi}\right), \tag{4.3}$$

into a fixed part using the background density $\rho_0 = 1$ and a fluctuating part with $\delta\rho = \rho - \rho_0$. While the basic idea of this technique was first discovered in the work of Skordos [141], it is applied in its most general form in LBMPY [1]. This is possible because the decomposed form of the equilibrium is directly used to derive the discretised update equations. Additionally, back transformations that are needed in special cases are inserted automatically. For example, the cumulants are not defined for $\rho = 0$ due to the usage of the logarithm. Thus, a back transformation must be applied before the collision step. Next, the collision space is specified by choosing a type of statistical quantity and defining its basis as a sequence of polynomials. Additionally, corresponding relaxation rates must be specified. Each relaxation rate can be provided either as a fixed numerical value or in symbolic form, allowing its value to remain undetermined until the runtime of the generated code. Especially when regularisation is used (e.g. higher order relaxation rates are set to unity), it is possible to simplify the resulting equations drastically. More details will be given later.

The final three components of the definition require a significantly more elaborate structure. To model equilibrium distributions, compute macroscopic quantities, and define force models, LBMPY provides specific hierarchies of Python classes. Abstract base classes define their respective interfaces. These components not only encapsulate information about the method but also play an active role during the code generation process.

For instance, the equilibrium object must produce an algebraic form (discrete or continuous) of its distribution, distinguish between fixed and fluctuating forms, and provide the background distribution in the latter case. Furthermore, it must provide methods to compute its raw moments, central moments, and cumulants. These methods are invoked during the symbolic derivation of the collision rule. The same applies to force models and the computation of conserved quantities; both components will contribute to the equations that make up the collision rule.

In addition to functional requirements, using classes offers significant advantages. While the most common components, such as the Maxwellian equilibrium or the Guo [80] force transformation, are already implemented in the current version of LBMPY, this structure makes LBMPY flexible and extensible. Developers can use the interfaces of the respective base classes to implement custom equilibrium distributions, force transformations, or procedures for computing macroscopic quantities.

4.3.3 Collision Rule

The next abstraction layer of LBMPY derives the equations of the collision rule from the abstract definition. Depending on the combination of storage and equilibrium format, an implementation for Equation (2.6) is derived in symbolic form. By manipulating the collision equations at the mathematical level, it is possible to leverage all available information about the method to simplify and optimise the equations. The derivation system is modular, combining equations generated by several components. These components

include the equilibrium and force model instances, which provide algebraic expressions of their respective representations in the given collision space, and the conserved quantity computation, which produces equations for density (or its analogues) and velocity (see Equation (2.11)) [1].

The equations forming the collision space transformation $\mathcal{T}$ are provided by a set of dedicated classes within LBMPY. The transformations to the moment- and central-moment space are linear. Thus, they can be expressed as matrix-vector multiplications. While this approach is simple, a more sophisticated idea is based on the so-called Chimera transform [49]. Raw moments are first computed from pre-collision populations to obtain monomial moments. Commonly, the moment space is spanned by polynomial expressions. Thus, a recombination of the polynomial space is applied afterwards. The Chimera transformation obtains its name from the fact that temporary quantities $m_{x|\beta\gamma}$ and $m_{xy|\gamma}$ are created that are neither populations nor moments but a combination of both. The final form implemented in LBMPY reads

$$\begin{aligned} f_{xyz} &:= \begin{cases} f_i & \text{if } \boldsymbol{\xi}_i = (x,y,z)^T \text{ is contained in stencil} \\ 0 & \text{otherwise} \end{cases} \\ m_{xy|\gamma} &:= \sum_{z\in\{-1,0,1\}} f_{xyz} \cdot z^\gamma \\ m_{x|\beta\gamma} &:= \sum_{y\in\{-1,0,1\}} m_{xy|\gamma} \cdot y^\beta \\ m_{\alpha\beta\gamma} &:= \sum_{x\in\{-1,0,1\}} m_{x|\beta\gamma} \cdot x^\alpha . \end{aligned} \tag{4.4}$$

The recursive nature of the Chimera transform minimises the number of arithmetic operations since each possible combination of populations and velocities is evaluated exactly once. For the backward transformation, first raw moments are created by decomposing the polynomial moments. Then a symbolic matrix-vector multiplication is applied $\boldsymbol{f}^* = \boldsymbol{M}^{-1}\boldsymbol{m}^*$. Those equations can be simplified by splitting them into their symmetric and antisymmetric part:

$$f_i^* = f_i^+ + f_i^-, \qquad f_{\bar{i}}^* = f_i^+ - f_i^- . \tag{4.5}$$

This split roughly cuts the number of arithmetic operations in half.

As shown in [1], monomial raw and central moments are bidirectionally related through binomial expansions

$$\kappa_{\alpha,\beta,\gamma} = \sum_{a,b,c=0}^{\alpha,\beta,\gamma} \binom{\alpha}{a}\binom{\beta}{b}\binom{\gamma}{c} (-u_x)^{\alpha-a}(-u_y)^{\beta-b}(-u_z)^{\gamma-c} m_{abc}, \tag{4.6a}$$

$$m_{\alpha,\beta,\gamma} = \sum_{a,b,c=0}^{\alpha,\beta,\gamma} \binom{\alpha}{a}\binom{\beta}{b}\binom{\gamma}{c} u_x^{\alpha-a} u_y^{\beta-b} u_z^{\gamma-c} \kappa_{abc}. \tag{4.6b}$$

This gives rise to a binomial Chimera transform that can transform raw moments into central moments. The binomial Chimera transform takes the form

$$\begin{aligned}
\kappa_{ab|\gamma} &:= \sum_{c=0}^{\gamma} \binom{\gamma}{c} (-u_z)^{\gamma-c} m_{abc} & m^*_{ab|\gamma} &:= \sum_{c=0}^{\gamma} \binom{\gamma}{c} u_z^{\gamma-c} \kappa^*_{abc} \\
\kappa_{a|\beta\gamma} &:= \sum_{b=0}^{\beta} \binom{\beta}{b} (-u_y)^{\beta-b} \kappa_{ab|\gamma} & m^*_{a|\beta\gamma} &:= \sum_{b=0}^{\beta} \binom{\beta}{b} u_y^{\beta-b} m^*_{ab|\gamma} \\
\kappa_{\alpha\beta\gamma} &:= \sum_{a=0}^{\alpha} \binom{\alpha}{a} (-u_x)^{\alpha-a} \kappa_{a|\beta\gamma} & m^*_{\alpha\beta\gamma} &:= \sum_{a=0}^{\alpha} \binom{\alpha}{a} u_x^{\alpha-a} m^*_{a|\beta\gamma}
\end{aligned} \tag{4.7}$$

Since each combination of moments and velocities is evaluated exactly once, the resulting expressions require minimal arithmetic operations. Polynomial central moments are constructed from monomial quantities after the forward transform and are decomposed before the backward transform.

As shown in Section 2.2, cumulants share a non-linear relationship with other statistical observers. This is because the logarithm is used in their expression. Cumulants are derived from central raw moments in LBMPY using the following bidirectional relation

$$C(\Xi) = (\Xi \cdot \boldsymbol{u}) + \log K(\Xi) \quad \Leftrightarrow \quad K(\Xi) = \exp\left(C(\Xi) - \Xi \cdot \boldsymbol{u}\right). \tag{4.8}$$

While this gives valid numerical equations, it also leads to the undesired situation in which transcendental mathematical functions are encoded in these equations. Such complex functions inside a compute kernel might lead to a severe performance decrease. Since logarithmic expressions are only associated with zeroth-order cumulants, it is possible to employ *SymPy* to eliminate them globally. This comes from lower-order central moments and cumulants being the same. Such knowledge is directly contained in LBMPY to leverage the simplification process.

To showcase the importance of the transformation process, a comparison of different commonly used techniques is shown in Table 4.1. Herein, the number of FLOPs is counted [5] for a set of populations shifted to the central moment space and back to the populations. Thus, Table 4.1 does not include a collision step. As pointed out before, the linear relationship between populations and central moments can be expressed in a single matrix. However, this matrix is generally dense, resulting in a significant amount of FLOPs. To overcome this problem, a common strategy is to split the transformation into the raw and central moments in two matrices. The second matrix (often called shift matrix) is lower triangular, and the first matrix is drastically sparser [53, 54, 142]. This approach's effectiveness can generally be confirmed by LBMPY. A fast central moment transform was proposed in the work of Geier *et al.* [49] for the D3Q27 stencil. In the scope of this thesis, the fast central moment transformation was generalised to arbitrary stencils. Due to its design, however, the fast central moment transformation is most effective for the D3Q27 stencil. Nevertheless, the newly developed binomial Chimera transformation significantly outperforms all other approaches in every case shown in Table 4.1. This result also strongly highlights that *SymPy* can not optimise arbitrary sets of equations. Especially on a high number of operations, it eventually fails to simplify. Thus, theoretical mathematical work is still required to use the computer algebra system effectively.

[5] Counting the operations is done on the symbolic equations within PYSTENCILS. A later compiler stage might decrease the number of FLOPs. However, from the author's experience, the compiler stage generally does not change the overall picture.

Table 4.1: Arithmetic operation counts in the symbolic representation of the central moment transformation. The central moment transformation is done using a full matrix (Matrix), two split matrices (Shift Matrix), the fast central moment transformation (Fast Transform) or the binomial Chimera transformation (Binomial Chimera). For each stencil, the forward and the backward transformation is shown separately, and the lowest number of FLOPs is marked using a bold font.

Stencil	Transform	Operation Counts	
		Forward	Backward
D2Q9	Matrix	626	659
	Shift Matrix	127	99
	Fast Transform	128	174
	Binomial Chimera	**88**	**87**
D3Q19	Matrix	4121	1998
	Shift Matrix	376	275
	Fast Transform	430	613
	Binomial Chimera	**233**	**239**
D3Q27	Matrix	12706	13111
	Shift Matrix	911	774
	Fast Transform	587	785
	Binomial Chimera	**402**	**409**

The transformation process already shows the great potential of the modularity offered by LBMPY. As a next step, the full collision operator is created. To do so, besides the transformation, the collision itself needs to be added, as well as additional equations like the calculation of the conserved quantities. After putting all ingredients together, LBMPY offers a powerful set of simplifications to optimise the final equations. In the following, the most important simplifications are listed briefly:

Conserved Quantity Rewriting Zeroth- and first-order raw moments are similar to the conserved quantities. Thus, equations computing ρ, $\boldsymbol{u}$, etc. from populations are directly replaced by their raw moments.

Collapsing Conserved Central Moments Equations for zeroth- and first-order central moments, e.g., obtaining, $\kappa_{000} = \rho$ and $\kappa_{100} = -F_x/2$ are collapsed. The underlying idea is that these equations can be removed completely because they are not influenced by the collision. Hence, the pre-collision expression can be inserted directly.

Propagation of Logarithms Cumulant collision models which still contain logarithmic expressions are undesired. Therefore, logarithmic and exponential expressions are propagated. This cancels both functions and leads to simplifications.

Common Subexpression Elimination *SymPy* offers a powerful CSE which was already shown in the work of Bauer *et al.* [26].

Expression Propagation It has proven advantageous to propagate some expressions directly where they are used. This includes assignments whose right-hand sides are constant, single symbols, products of macroscopic quantities, or multiples of body force components.

All the simplifications used in LBMPY are especially effective because *SymPy* automatically tries to simplify expressions whenever they are manipulated, for example, by replacing symbols with other expressions. Thus, every propagation step also triggers a simplification step, which eventually causes a collapse of overall mathematical terms. An important flavour of the simplification routines is that variables invariant of the collision process are directly propagated to the backward transformation step. To illustrate this, a simple example should be given. Without the propagation step, the binomial Chimera transform would include assignments similar to

$$\begin{aligned} \kappa_{000} &= \rho \\ \kappa_{100} &= -\frac{F_x}{2} \\ \kappa^*_{000} &= \kappa_{000} \\ \kappa^*_{100} &= \kappa_{100} + F_x \\ m^*_{10|0} &= \kappa^*_{000} u_x + \kappa^*_{100}. \end{aligned} \tag{4.9}$$

The propagation steps described above will cause most of these assignments to be dropped. The only remaining equation is

$$m^*_{10|0} = \rho u_x + \frac{F_x}{2}. \tag{4.10}$$

The most significant advantage that this toolchain, with its simplification, offers is that every information provided by an end user is directly used in the derivation process. An especially effective algebraic simplification occurs when relaxation rates are set to unity. In this case, propagation eliminates large portions of the forward collision space transformation and significantly simplifies the backward transformation. Substituting $\omega = 1$, a relaxation equation $q^*_p = q_p + \omega\left(q^{\text{eq}}_p - q_p\right)$ is immediately simplified to $q^*_p = q^{\text{eq}}_p$. In this case, the forward transformation is no longer required and can be eliminated entirely. A collection of collision models with different optimisations and lattice stencils is presented in Table 4.2. Their arithmetic operation is counted similarly to the previous section. Besides the more simple single-relaxation-time (SRT) and two-relaxation-time (TRT) methods, also multiple-relaxation-time (MRT), weighted multiple-relaxation-time (WMRT), central-moment (C) and cumulant (K) based methods are shown. All methods are also shown in a regularised version (see Section 2.2.2 indicated by the prefix 'R-'. Furthermore, the K17 cumulant method is added for the D3Q27 stencil (other stencils are not formulated for this optimisation). In all cases, the employed simplification together with *SymPy*'s CSE drastically reduces the overall FLOPs. This result is especially impressive because some collision operators, which are considered more stable and accurate in the literature, cause a lower computational footprint than simpler models. Especially the regularised WMRT method can profit significantly from the simplification toolchain provided by LBMPY. This method is designed so that higher-order equilibria are zero. Regularising these equilibria and applying propagation steps to the resulting equations causes a drastic reduction of FLOPs.

Table 4.2: Arithmetic operation counts in the symbolic representation of compute kernels generated by LBMPY for several method definitions on the D2Q9, D3Q19 and D3Q27 stencils. Numbers for kernels derived without simplification (N), standard simplification (S) and simplification plus CSE (S+CSE) are listed.

	D2Q9			D3Q19			D3Q27		
Method	N	S	S+CSE	N	S	S+CSE	N	S	S+CSE
SRT	448	113	**92**	1156	312	**207**	3842	428	**287**
TRT	437	191	**118**	1161	480	**257**	3847	668	**361**
O-MRT	196	142	**121**	554	397	**314**	928	596	**485**
R-MRT	196	105	**87**	554	290	**222**	928	407	**320**
WMRT	178	117	**102**	507	339	**282**	843	484	**411**
R-WMRT	178	87	**74**	507	244	**194**	843	316	**261**
C	236	156	**132**	638	415	**347**	1140	747	**605**
R-C	236	108	**95**	638	255	**219**	1140	417	**347**
K	676	167	**142**	1854	454	**379**	7623	1035	**824**
R-K	676	114	**99**	1854	272	**234**	7623	489	**402**
K17	-	-	-	-	-	-	7836	1033	**782**

4.3.4 Update Rule

The collision rule derived in the previous section constitutes the relaxation process for a single cell whose populations are represented purely symbolically. Up to this stage, LBMPY depends entirely on *SymPy* and can be seen as a package for the symbolic derivation of highly efficient LBM. As a final step, the substitution mechanisms of *SymPy* are used to replace the q symbolic populations with representations using the `Field.Access`-class of PYSTENCILS. As shown in Section 4.2, this allows the expression of neighbouring information in the form of index notation. The final touch to increase the generality of LBMPY is to support arbitrary streaming schemes for the LBM and PYSTENCILS fits this role perfectly. Common streaming schemes like the pull or push pattern can be added by creating a substitution dictionary mapping pre- and post-collision PDFs to the correct field neighbour accesses. The creation of these dictionaries is encapsulated in specific Python classes that use general mathematical rules to support arbitrary lattice stencils. Likewise, more complex streaming patterns like the AA-pattern [38], the Esoteric Twist [39], and the Esoteric Pull and Push scheme [40] are supported. Due to the generality of PYSTENCILS, it is possible to support entirely different streaming mechanisms. For example, an indirect streaming scheme was also implemented recently [9]. The big advantage of this approach is that the full collision toolchain can be combined with any of the mentioned schemes. After creating the full update rule, the code generation pipeline of PYSTENCILS is used to create executable compute kernels.

4.4 WALBERLA

As outlined in the previous sections, the code generation facility becomes especially powerful in combination with an existing high performance computing (HPC) framework. Herein, this thesis focuses on the WALBERLA framework. At its core, WALBERLA is a framework to parallelise stencil codes on regular cartesian grids. As the name WALBERLA (widely applicable Lattice Boltzmann from Erlangen) already indicates, it has a focus on the LBM. The basic core design of WALBERLA consists of supporting massively parallel applications. Thus, in general, no data structure exists that does not scale with the number of involved processes. The careful performance-driven software design leads to excellent scalability on recent supercomputers [22, 25, 26, 28, 4, 5]. The framework provides various different communication strategies, including synchronous and asynchronous ghost layer exchange via MPI.

WALBERLA is written in C++ which provides the necessary freedom to access memory efficiently. Currently, WALBERLA supports C++ 17 and can be compiled with all modern compilers without external dependencies. Even the almost necessary dependency to MPI comes as optional, and other features like GPGPU support, complex meshes or enhanced load balancing can be activated in the *CMake* [6] build system when optional dependencies are provided [24]. In massively parallel simulation, the probability of failure increases due to using several thousands of compute cores. For this reason, WALBERLA provides a mechanism for fault tolerance [143].

As highlighted in the previous sections, PYSTENCILS and LBMPY follow a software design concept that can be expressed in different layers which build on each other. Similarly, WALBERLA organises subsets of functionalities in so-called *modules*. Only the minimally necessary dependencies between the modules exist to increase the maintainability of the software. Simulations in WALBERLA are realised by writing apps which compose these modules. In this fashion, apps can be lightweight when only a certain aspect of the WALBERLA framework is needed. On the other side, users can extend modules inside their applications if special features are missing from the core modules.

While WALBERLA offers great maintainability on the general software level, it also needs to execute highly optimised compute kernels at the hotspot of numerical algorithms. As highlighted in the previous sections, these compute kernels must be adapted or rewritten to achieve the highest possible performance on specific hardware. Consequently, the great level of maintainability fails on the compute kernel level when specific optimisation techniques like SIMD vectorisation are introduced manually. It is exactly this part where code generation techniques significantly leverage the WALBERLA framework. WALBERLA already offers a high-level interface in the Python programming language for setting up and configuring simulations [144]. Therefore, PYSTENCILS as a DSL embedded in Python is an ideal candidate for kernel code generation without introducing new dependencies.

The following sections show the most important data structures and concepts of WALBERLA used in this thesis.

[6] https://cmake.org/

4.4.1 Related Work

Before going into details of the software structure of WALBERLA, we show some frameworks that share similar ideas to WALBERLA. This section is intended to review them and provide an overview of the state of the art. In the past decades, many frameworks have been developed for the LBM. Many of these packages have been gathered in a list published on GitHub [7]. In the following we discuss the most relevant frameworks.

OpenLB

The OpenLB package is a flexible simulation framework with the LBM on uniform grids [145, 146]. The code is written in C++ and parallel simulations are achieved using MPI. For the usage of NVIDIA GPGPUs compute unified device architecture (CUDA) is used directly. OpenLB implements the most common moment-based collision models and provides optimised implementations for each. The streaming step is solved using shift-swap-streaming by Kummerländer *et al.* [41] as detailed in Section 2.2.1. High-level optimisations are supported using a code generation pipeline that parses handwritten C++ code to Python, applies symbolic optimisations using *SymPy*, and parses the result back to be compiled within OpenLB.

Palabos

The Palabos (Parallel Lattice Boltzmann Solver) software library evolved from the OpenLB framework in an attempt to support more complex multiphysics applications [147]. Therefore, they share similar philosophies but differ vastly in the code basis. For example, Palabos supports mesh refinement and accelerators using the standard library of C++. Palabos probably has the widest range of supported collision models from a unified mathematical formulation that is provided as a code template within the software stack.

VirtualFluids

The VirtualFluids framework became available open-source recently [148]. The code base is focused on the cumulant collision operator, which is implemented in optimised versions using C++ and CUDA. Furthermore, mesh refinement is supported using the compact interpolation by Geier *et al.* [149]. A specialised version of the Esoteric Twist is used for indirect addressing for the streaming pattern. With pointer-chasing techniques, the memory overhead, which normally arises in indirectly addressed formats, is kept low.

Musubi

Musubi is a multi-level parallel LBM solver that is part of the APES (Adaptable Poly-Engineering Simulator) suite[8]. It was designed to handle complex simulations of incompressible fluid flows, particularly on a large scale, using an octree data structure to represent sparse meshes. In this way, the framework supports mesh refinement using the compact interpolation scheme [150]. The software stack is written in FORTRAN and does not support accelerators like GPGPUs. Highly optimised implementations for the most advanced collision operators are provided [65].

[7] https://github.com/sthavishtha/list-lattice-Boltzmann-codes
[8] https://geb.inf.tu-dresden.de/apes-suite/

FluidX3D

The FluidX3D framework is a recently published open-source lattice Boltzmann (LB) solver[9]. A notable feature of FluidX3D is its use of OpenCL 1.2[10], which ensures broad compatibility across all architectures supported by OpenCL. Additionally, the framework includes integrated in situ visualisation, enabling real-time observation of simulations. The code is also meticulously optimised for low memory usage through the use of in-place streaming patterns and single- or half-precision data types for storing the PDFs [40, 151]. However, at the time of writing, the framework has limited domain decomposition capabilities for running simulations on large supercomputers and supports only SRT-based collision operators. Moreover, mesh refinement is not yet supported.

TCLB

The TCLB framework is an actively developed open-source LB solver [152, 153]. It supports a large variety of models including multiphase, electrokinetic or reaction models. The idea behind TCLB is to describe the model in an abstract way using the R programming language. From this description, all compute kernels are generated and compiled. TCLB supports accelerators from NVIDIA and AMD using CUDA and HIP. With the usage of MPI, TCLB achieves large-scale simulations on uniform simulation domains.

4.4.2 The Block Structured Octree

The basis of WALBERLA is formed by a forest of octrees distributed to several processes. While the concept was shown in great detail in the work of Schornbaum *et al.* [23, 24], a rough overview should be given here since it forms the basis for further developments realised in this thesis. The creation of these octrees happens at the initialisation phase of a simulation. An overview of the initialisation phase is given in Figure 4.5. It shows the setup for a virtual wind tunnel containing a spherical object.

The first step of the initial phase is to create a domain axis-aligned bounding box (AABB) subdivided into one or more blocks. These blocks are called root blocks, and they form the basis of the simulation by creating the coarse grid. The most important concept is that each of these blocks is represented by a lightweight class that does not yet contain any heavy data. Instead, it only contains minimal information like its block AABB, an identification number and a corresponding MPI rank. With this minimal information, successive refinement steps are executed. A user-defined function is called in each refinement step, which evaluates if a block should or should not be refined. In the case presented here, this user-defined function would compare each block's AABB with the spherical object [11]. In case of an overlap, the block will be marked for refinement.

[9] https://github.com/ProjectPhysX/FluidX3D

[10] https://www.khronos.org/opencl/

[11] In the case of a simple shape, like in this example, a representation as a mathematical function is enough. For more complex cases, WALBERLA supports the STL file format to read mesh information

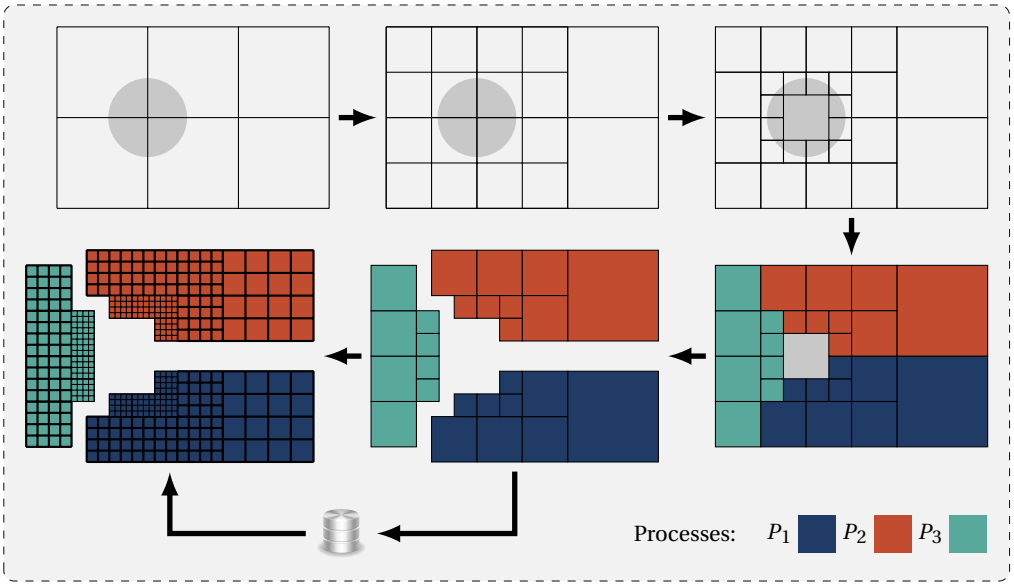


Figure 4.5: The domain partitioning process in WALBERLA for a virtual wind tunnel containing a spherical object. First, the domain bounding box is subdivided into cuboidal subdomains. These blocks can then be refined in a 2:1 ratio. If a block overlaps fully with a block, it is excluded from the setup. Afterwards, each block is distributed to a process (here shown with three processes) in a static load balancing step. The setup can be stored in a file to jump the initial phase and the load balancing a production run, or the run can be started directly. In each case, the previously created blocks are divided into a fixed number of cells on which further algorithms are executed.

Each of the marked blocks is subdivided into eight finer blocks. Like this, each root block forms an octree, which results in a forest of octrees globally. Due to enforcing a 2:1 refinement ratio, consecutive passes over the octrees are executed to refine additional blocks if needed. Increasing the number of refinements eventually results in whole blocks overlapping with the object. Thus, in this thesis, the possibility is implemented to detect these situations and exclude these blocks from the domain. In the work of Schornbaum *et al.* [23, 24], this was only possible for root blocks.

Once the refinement step is completed, the blocks are distributed to processes in the following load-balancing step. Specialised load-balancing strategies based on space-filling curves are provided by WALBERLA, but also an interface to the ParMETIS [12] library is offered for general load-balancing algorithms. As indicated in Figure 4.5, commonly, a level-wise balancing is applied in the case of setups for the LBM. This is because the most used algorithms for the LBM on non-uniform grids work with spatial and temporal refinement [23]. This means that cells on a finer level are executed more frequently. Hence, the workload on fine blocks is significantly higher, which needs to be respected by the load balancer. More information will be given in Chapter 5. A more specialised workload assignment is possible by providing callback functions that return the workload for each block based on user-defined criteria.

[12] http://glaros.dtc.umn.edu/gkhome/views/metis/parmetis/

At this point, each block is associated with a valid MPI rank and holds a link to every neighbouring block. However, the whole domain decomposition is still physically located in a single process. From this standpoint, the block decomposition can be stored in a file, and later, the setup can be reloaded from it to jump the initialisation phase in a production run. This is especially useful in extreme scale scenarios where the load balancing would cause a significant overhead for the overall runtime.

As a next step, the forest of octrees is actually distributed to the target processes. Each of the processes then allocates the heavy data. It results that from this point on, the blocks are divided into a fixed number of cells, which are defined beforehand. Their usage switches now to containers for arbitrary simulation data. Most importantly, arrays can be created for each block, which stores the actual simulation data. More information on WALBERLA's array structures is presented in the following section.

The octree-based decomposition presented here holds a block in each leaf. Each of these blocks eventually contains the actual simulation cells. As indicated in Figure 4.5, this results in a certain overlap between the geometry and blocks at its surface. Alternatively, each block could hold only a single cell instead of a group. In this case, only a single layer of boundary cells would be created, which would be the minimal case. This approach is often referred to as cell-based mesh refinement. While this approach is maximally flexible, it comes with several drawbacks.

1. The resulting octree will be much larger. For example, when considering a block with $4 \times 4 \times 4$ cells (as shown in Figure 4.5), the octree size is almost two orders of magnitude smaller in a three-dimensional setup. Thus, the whole setup phase, including the load balancing, is significantly faster, which opens the door to dynamic load balancing and thus adaptive mesh refinement (AMR) as shown by Florian Schornbaum [29]. Also, strategies like storing the decomposed octree in a file would be less effective.

2. Modern architectures favour consecutive memory access. Organising the data in small chunks naturally causes chunks of consecutive memory, which can be accessed most efficiently.

3. In LBM simulations typically links to all nearest neighbours are needed. In the case of a block-structured mesh, the neighbouring cells can be obtained from the loop counters over the cells inside a block. Only at block borders neighbouring information must be extracted from the octree. Cell-based mesh refinement might introduce an overhead here because the neighbouring information must be extracted from the octree for every lattice cell.

Lastly, the natural question arises if a cell-based decomposition's additional flexibility is needed in complex applications. If this is true, the mentioned disadvantages must be accepted. However, looking at complex industrial applications shows that grid transitions of a single cell are not feasible [154]. For example, a smooth interface is created by using four cells between grid transitions in the thesis of Andrea Pasquali [155] for the aeroacoustic simulation of organ pipes. Thus, it is favourable to encode this constraint in the software design and use it advantageously.

4.4.3 Fields and Sweeps

As outlined before, a WALBERLA `block` forms a container for arbitrary data. The most prominent data structure of WALBERLA, which can be stored in a block container, is the `Field`-class and its subclasses. `Fields` are cuboid d-dimensional data structures stored as contiguous linearised arrays. An arbitrary number of datums can be stored for each coordinate (x, y, z). Thus, in general, WALBERLA is designed to express problems within this concept. With high-level methods, single values can be extracted by providing their location with a 4-tuple: `Field::get(x,y,z,i)`. For GPGPU simulations, the core data structure is the `GpuField`, which is a C++ class used from CPU code that manages pointers and indexing information of a GPGPU array holding the actual data. Only NVIDIA GPGPUs were originally supported by WALBERLA. Due to the increasing importance of AMD GPGPUs for HPC (at the time of writing, the number one supercomputer in the TOP500 [13] was based on AMD GPGPUs) a porting to this architecture was realised within this thesis.

Since the data is encapsulated within WALBERLA's Octree structure, a simple and intuitive method is essential for formulating algorithms. To achieve this, WALBERLA utilizes the concept of `Sweeps`. An example implementation is illustrated in listing 4.5 to provide a clearer understanding, but a more comprehensive explanation can be found here [156]. A `Sweep` is a callable C++ function that takes a `Block` pointer as its only input parameter. Within the function, data is extracted from the `Block` using a `BlockDataID`, which serves as an input to its `getData` method. The `BlockDataID` assigns a unique identification number to each data object on the block structure. Due to its design, additional information is encoded in the `BlockDataID`, described in detail here [29]. Once the data is accessed, various update rules can be applied to it. Subsequently, these `Sweeps` are integrated into a time-stepping function that executes them across the `BlockForest`.

CODE LISTING 4.5: The `Sweep` concept in WALBERLA forms the basis to formulate algorithms. At its core, a `Sweep` is a callable, receiving a pointer to a block. From the block, data is extracted, and an algorithm is executed. In this example, the `Sweep` is realised as `Functor` class, but other implementations are possible.

```
class ExampleSweep {
   ExampleSweep( BlockDataID fieldID ) : fieldID_( fieldID ) {}

   void operator()( Block * block ) {
      // get data pointer from block structure
      auto field = block->getData< Field<double, 1> >( fieldID_ );

      // implementation of some update rule
      for( auto& iter : *field ) {
         iter = ...;
      }
   }

   BlockDataID fieldID_;
};
```

[13] https://top500.org/lists/top500/list/2024/06/

4.4.4 The Communication Infrastructure

In the context of WALBERLA, communication refers to a data exchange between two blocks inside the octree. These may or may not share the same MPI rank. So-called communication schemes are provided to manage the data exchange between the blocks. While communication schemes differ in detail, they have in common that they all traverse the Octree and identify regions for data exchange between each block pair. For each region, functions are called to execute the data exchange. These functions are encapsulated using a common interface, which is referred to as pack info. The interface in a simplified version is provided in listing 4.6 to give a better understanding. This workflow completely separates the logic of which data needs to be exchanged from the actual execution that exchanges this data most efficiently. For intra-process communication (i.e. both blocks share the same MPI rank), pointers to the data arrays serve as input parameters (these pointers are extracted from the corresponding blocks), and the exchange can be performed in place. On the other hand, to minimise costly calls to MPI communication routines, a buffered communication system is employed in the case of inter-process communication. Herein, the communication scheme manages two buffers (for each process pair), one for sending and one for receiving data. By calling the corresponding functions from the pack info, data is packed into the sending buffer on the source process and unpacked on the destination process after the MPI communication.

CODE LISTING 4.6: Simplified interface for uniform pack info class. The subclasses must implement the interface methods.

```
class PackInfo {
   // pack data in buffer array
   void packData( Block * sender, stencilDirection dir, SendBuffer & buffer );
   // unpack data from buffer array after the MPI communication
   void unpackData( Block * receiver, stencilDirection dir, RecvBuffer & buffer );
   // communicate data between blocks on the same process
   void communicateLocal( Block * sender, Block * receiver, stencilDirection dir );
};
```

The exemplary interface in listing 4.6 provides two functions for the data exchange between two blocks on different MPI ranks and one function for blocks on the same MPI rank. The `packData`-function receives a pointer to the sending block from which it extracts pointers to the actual data arrays. These pointers are handed to the communication kernels, which copy data from them to the buffer array. Commonly in WALBERLA, data exchange is performed direction-wise, which is realised through the stencil direction that is an argument of the `packData`-function. Especially for the LBM this allows for important optimisations. The data exchange between four blocks is illustrated in Figure 4.6 exemplary. In this two-dimensional example, the data that needs to be shared between the blocks differs in each direction. As an alternative to this specialised direction-wise treatment, the ghost layer of the receiving blocks could be filled entirely. This has two disadvantages. First, it would cause an unnecessary communication overhead, and second, this variant might not be compatible with a certain streaming pattern. Realising a specialised direction-wise communication pattern can quickly become hard to maintain. For this case, the code generation facility is ideal, as it allows the generation of specialised communication kernels for each direction, which the pack info will execute.

The counterpart of the `packData`-function is the `unpackData`-function. Both functions share a similar interface, however, from the perspective of the receiving block. Hench, a pointer to the receiving block is provided from which pointers to the actual data arrays are extracted. This time, the compute kernels read data from the buffer array and store it on the extracted data pointers.

The last function `communicateLocal` can be seen as a combination of the previous function. It receives pointers to the sender and receiver block, which is possible because they reside on the same MPI rank. From the block pointers, it extracts pointers to the underlying data arrays and performs a data exchange between them with respect to the lattice direction.

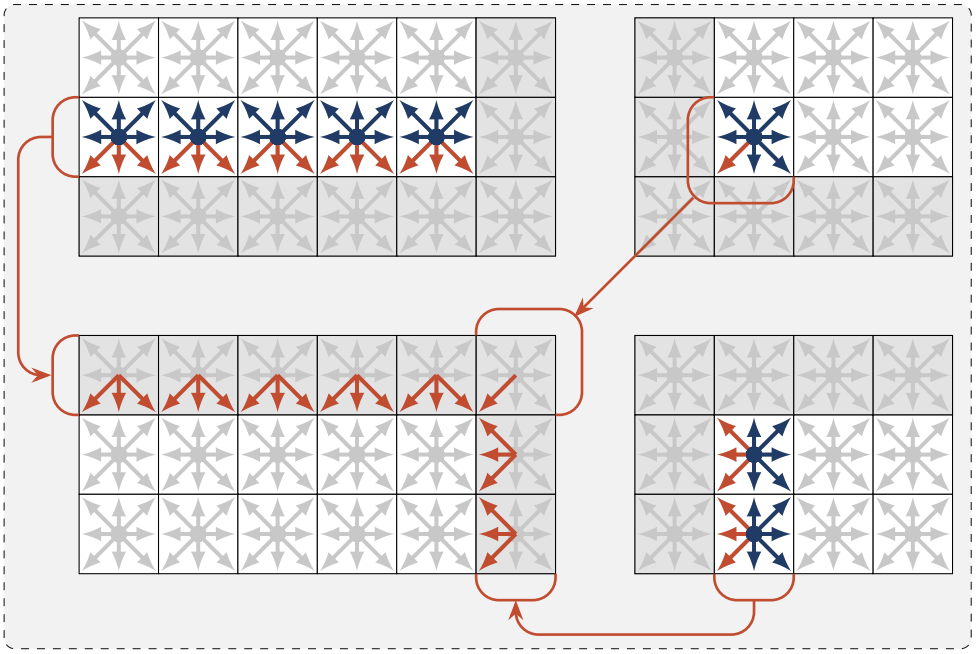


Figure 4.6: Data exchange between four blocks which have the same spatial resolution. The example shows a two-dimensional grid that stores the PDF array of an LBM simulation. The ghost layers around each array are marked in grey, and the data PDF directions that need to be communicated are shown in red. For the sake of simplicity, only the communication to the block in the lower left corner is shown, and the pull stream pattern is assumed.

4.4.5 Integration of the Code Generation

The code generation approach with PYSTENCILS forms a powerful tool for rapid development due to its high flexibility, interactive development environment using *IPython* [14], comprehensive mathematical optimisations and architecture-specific low-level code adaptations. However, the PYSTENCILS package lacks essential features for simulating relevant real-world problems. The most prominent missing feature is the lack of inter-node communication. While PYSTENCILS can use full processors using OpenMP directives,

[14] `https://ipython.readthedocs.io/en/stable/`

it is limited to using a single process. Furthermore, it has no notion of grid hierarchies, which makes any form of mesh refinement hard to realise in PYSTENCILS itself. Other features building on these would be the ability to handle complex geometries (e.g. given as triangular mesh files) or load balancing.

As PYSTENCILS generates low-level C-code and uses a clearly defined API based on C-style pointers, it is by design possible to integrate it with existing C++ frameworks. However, as the previous sections show, high-level classes are usually employed in large frameworks to hide low-level complexity from end users. The WALBERLA framework is no exception here. Thus, wrapper code is necessary to integrate generated code parts. As this wrapper code needs to be flexible as well because it needs to be adapted to any generated code, a simple but powerful template engine is used with the *Jinja* package [15]. The *Jinja* package is also used by Eibl *et al.* [157] to integrate the *Modular and Extensible Software Architecture for Particle Dynamics* (MESA-PD) framework for particle dynamics in WALBERLA. Thus, the workflow employed here shares similarities with already existing design decisions and does not add any additional dependencies. An overview illustration of the combination of PYSTENCILS and WALBERLA is given with Figure 4.7.

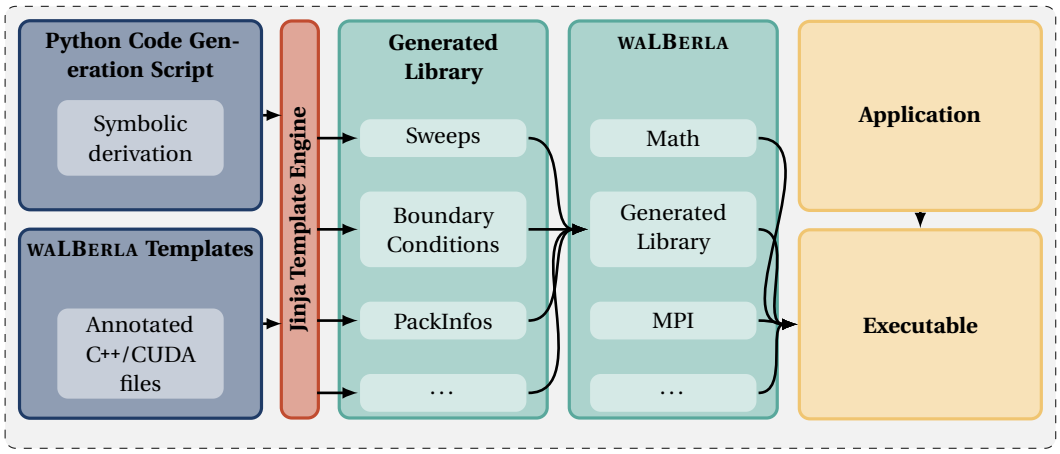


Figure 4.7: Integration of code generation in WALBERLA. A Python script using PYSTENCILS generates compute kernels. These are printed in template files using the *Jinja* package. The generated files can then be used as a new module within WALBERLA. The figure is inspired by the work of Eibl *et al.* [157] as *Jinja* templates are also used to integrate MESA-PD into WALBERLA

The PYSTENCILS package can be used standalone to solve problems fitting for a single process. Employing GPGPUs already gives enough computing power to run smaller problems and especially to carry out proof of concepts for newly developed models. The integration with WALBERLA picks up from this basis. Thus, the basic idea is to fully use the existing code generation workflow without changes until the actual compute kernels are generated. This is highlighted in listing 4.7, which shows a basic Python script to generate all necessary ingredients to solve Equation (4.1). The first part of the code generation script is exactly the same when just PYSTENCILS is used without integration into WALBERLA. With `pystencils_walberla`, a collection of Python functions is provided by WALBERLA for the integration, where the most important class `CodeGeneration` provides a context to control the code generation process from PYSTENCILS. This context is tightly coupled

[15] https://jinja.palletsprojects.com/en/3.1.x/

to the build system of WALBERLA and knows some important build information like using accelerators or shared memory parallelism. In general, all information from the building process could be extracted and provided to PYSTENCILS if needed. The `generate_sweep` function generates the main compute kernel, fitting to the actual build information. However, the finished compute kernel will be provided to *Jinja* templates to be embedded in a `Sweep` template class. Since *Jinja* is a Python package, this template can be enhanced with Python functions to extract the necessary information from PYSTENCILS. This information mostly concerns the data type information which is needed to extract `Field` pointers from WALBERLAs `BlockForest`. Shape and stride information is extracted from the `Fields` and provided to the compute kernel at runtime.

CODE LISTING 4.7: Example Python script to generate a WALBERLA module for solving the ADE (see Equation (4.1)). The `CodeGeneration` forms a context tightly coupled with the build system of WALBERLA. Therefore, information can be extracted to generate build-specific compute kernels.

```
import sympy as sp
import pystencils as ps
import pystencils_walberla as psw

dim = 2
c, c_tmp, u = ps.fields(f"c, c_tmp, u({dim}): [{dim}d]")
D, dx, dt = sp.symbols("D dx dt")

pde = ps.fd.transient(c) - ps.fd.diffusion(c, D) + ps.fd.advection(c, u)
discretize = ps.fd.Discretization2ndOrder(dx, dt)
assign = ps.Assignment(c_tmp.center(), discretize(pde))

dirichlet = ps.Dirichlet(sp.Symbol("concentration"))
neumann = ps.Neumann()

with psw.CodeGeneration() as ctx:
    target = ps.Target.GPU if ctx.gpu else ps.Target.CPU
    psw.generate_sweep(ctx, 'ADE', assign, field_swaps=[(c, c_tmp)], target=target)

    psw.generate_boundary(ctx, 'Dirichlet', dirichlet, field=c, target=target)
    psw.generate_boundary(ctx, 'Neumann', neumann, field=c, target=target)

    psw.generate_pack_info_from_kernel(ctx, 'ADEPackInfo', assign, target=target)
```

Besides the actual compute kernel, boundary conditions are necessary to define any problem fully. These boundary conditions are generated with the `generate_boundary` function. It gets a boundary object provided by PYSTENCILS together with the field on which the boundary condition is applied. Since boundary conditions are only applied on parts of the domain, integrating them in HPC software can be inherently difficult. An idea for applying boundary conditions is to use a flag field indicating which boundary condition is used at each cell. Then, traversing this flag field, the boundary condition is applied according to the boundary flag of the cell. Based on the thesis of Christian Feichtinger [156], a boundary handling class has been developed in WALBERLA, which receives boundary conditions as template arguments. Each boundary condition comes from a defined interface which implements how the cell is manipulated when a boundary condition is found. Using variadic templates allows for an arbitrary number of boundary conditions at compile time, and costly function pointers at runtime can be avoided. However, the resulting code is highly complex and hard for new developers to understand.

Additionally, correctly initiating the boundary handling class with the template parameters can be tedious and lead to only hardly readable error messages when done wrong. Lastly, Christian Feichtinger had to provide a separate handling class to enable boundary conditions on GPGPUs.

All the mentioned problems are avoided entirely with the code generation facility. The reason is that boundary conditions are now tailored to the simulation setup. Thus, they are generated with their own management structure (encoded in the *Jinja* template) to integrate a globally defined flag field. From the flag field, the coordinates of boundary cells are extracted in a setup step and stored into a simple standard vector. At runtime, this vector is traversed. It removes indirections entirely and enables GPGPU support seamlessly because the index vector can be copied to the device memory in the same setup step, while the actual update rule of the boundary condition is generated architecture specific by PYSTENCILS.

The last important ingredient for large-scale simulations is inter-process communication. In WALBERLA, this is realised by a separation of code and data using encapsulated kernels inside a common interface called pack info. The idea is detailed in Section 4.4.4. The pack info class can be generated directly using the function `generate_pack_info_from_kernel` (compare listing 4.7). Here, the strength of PYSTENCILS lies in the fact that all array accesses are available in a high-level description within the AST. Thus, routines to pack and unpack MPI buffers can be generated directly from the `Assignements` of the compute kernel. This enables three big advantages:

1. The packing and unpacking kernels can be generated specifically for some target hardware, which enables accelerator support seamlessly.
2. Several arrays can be packed and unpacked in the same kernel using the same buffer. For example, in the case of the ADE, values for the concentration and the velocity array have to be communicated (see Equation (4.2)). Normally, this would be done in separate packing routines for the sake of readability and flexibility.
3. Ghost layers do not always need to be communicated fully. Instead, the only necessary values per direction are communicated. This is important for arrays with multiple values per cell where only specific values are read and written in each direction. A prominent example would be LB streaming routines.

After generating the wrapper classes for WALBERLA, these are combined into a generated library and provided as additional modules to the WALBERLA software stack. In this workflow, the original code base can be used to its full potential, and only flexible parts or hotspot compute kernels that need a tremendous amount of optimisation are added as needed. In the final application, the generated library is included in the same way as any static module. Due to the full integration of WALBERLA's build system, no separate code generation step is needed. Instead, the code generation becomes part of the building processes, and generated files are compiled on the fly after their generation.

4.4.6 Extensions for the LBM

The LBM has been a core part of WALBERLA since its creation. For this reason, sophisticated infrastructure exists to express various complex collision models along with forcing schemes, boundary conditions or turbulence models. Herein, the API is designed to create a modular design but also to provide a simple interface that functions as a building block

for any extension. Recently, this infrastructure has been used to realise particle flows in a free surface river bed [158], which highlights its flexibility. In the following, a short overview of the existing infrastructure is given, followed by an overview of the newly developed *lbm* module.

Existing Approach

Many different choices exist to apply the LBM. The most significant differences lie in the choice of the collision operator, but also other parts like the force transformation exist in different versions as outlined in Section 2.2. On the other side, populations themselves are physically meaningless, and information needs to be extracted using statistical quantities like moments. Furthermore, architecture-specific optimisations must be applied to hotspots like the stream-collide step to achieve maximum performance. A sophisticated infrastructure has been built in WALBERLA using static polymorphism in the form C++ templates to account for these requirements. An overview of the most important classes is given in Figure 4.8. At the centre of the `lbm` module is the `latticeModel`. The `latticeModel` forms an interface to define the basic ingredients of a LB simulation. This concerns, e.g., the lattice stencil, the lattice weights or the collision and force model.

An important design choice is that the `latticeModel` forms a template parameter for the basic data structure, which is called `pdfField`. Since the `latticeModel` has all the information that defines the used LBM, it can directly access macroscopic information like velocity or density from its underlying populations. The `Sweep` for the stream-collide algorithm is chosen from several specialised implementations based on the `latticeModel`.

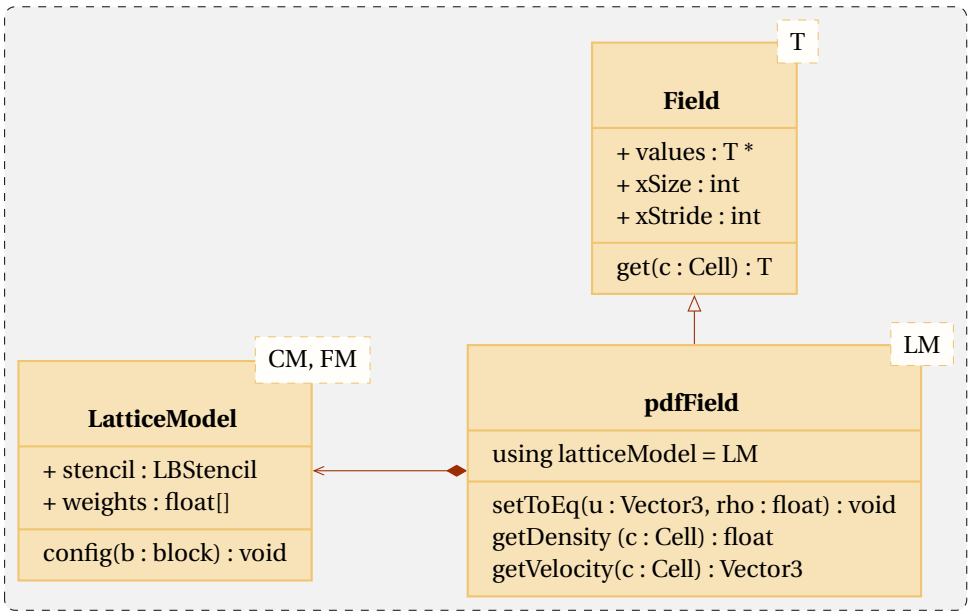


Figure 4.8: The existing *lbm* module of WALBERLA relise on static polymorphism in the form of C++ templates to provide a flexible implementation of different LBMs. The `latticeModel` defines all needed ingredients and receives the collision model (CM) and force model (FM) as template parameters. The `pdfField` forms the data structure to store the PDFs and receives the `latticeModel` as a template parameter. With the `latticeModel`, suitable functions are chosen to extract macroscopic parameters from the population array.

This approach has proven extremely useful since it allows the compiler to resolve all needed ingredients and provides the user with high flexibility because they are separated. Furthermore, combining the data structure with algorithmic knowledge allows a simple combination of other algorithms and, thus, opens the door for high-performant multiphysics simulations. On the other hand, numerous disadvantages arise:

1. Many specialised collision model implementations must be developed and maintained.
2. Generally, data structures and algorithmic logic are fully separated in WALBERLA. The advantage of this is that the data structure can easily be transferred to targeted hardware, and specialised target implementations can be executed on it. However, the tight bond of the algorithmic logic formed by the `latticeModel` and the `pdfField` violates this concept.
3. The `pdfField` relies on C++ cellwise operations to extract information from the populations. Cellwise operations can cause performance issues on accelerators like GPGPUs when these operations concern only a small number of cells.
4. The approach supports only the pull streaming pattern. Supporting other streaming patterns would be hard to realise because specialised implementations must also be provided.

The code generation integration from Section 4.4.5 can solve the first problem directly by providing a *Jinja* template of a `latticeModel`. Since the `LBMethod` of LBMPY has similar knowledge than WALBERLA's `latticeModel`, it is possible to generate a specialised highly optimised `CollisionModel` and extend it with method-specific accessor for the macroscopic variables. This idea is rather simple and has the advantage that the existing infrastructure of WALBERLA's *lbm* module can still be used fully along with the enhanced flexibility and optimisations provided by PYSTENCILS and LBMPY. However, the problem of lacking accelerator support can't be solved in this way. Therefore, in the scope of this thesis, a new design has been developed to combine the advantages of the old *lbm* module with the code generation facility.

New Approach

Building the old *lbm* module of WALBERLA, a new module has been created that fully incorporates the code generation approach. An overview of the most important classes is given with Figure 4.9. Like the `latticeModel`, the `LatticeStorageSpecification` holds essential information about the used LBM. However, all of these are generated in place from the symbolic description provided by LBMPY. This means that the `LatticeStorageSpecification` serves a rather descriptive purpose. The idea is to bind this class to the `pdfField` using static polymorphism. In this way, algorithm developers can use `pdfField` to correctly obtain needed information of the LBM. Additionally, the packing and unpacking kernels for the PDFs are incorporated with the `LatticeStorageSpecification`. Hence, the data structure knows how to communicate its datums with neighbouring processes. As a result, it is the only input for a pack info class that uses the compute kernels to execute the packing and unpacking of data. Furthermore, the `LatticeStorageSpecification` holds an `Accessor` struct, which gives the `pdfField` knowledge about the employed streaming pattern and allows to access

pre- and post-collision PDFs correctly. Since the data structure is lightweight, a simple equivalent can be implemented for supporting GPGPUs. Alternatively, it would be possible to incorporate the device support directly in the `pdfField` using unified memory or manually implementing synchronisation functions. Here, however, a second class is used for explicitness.

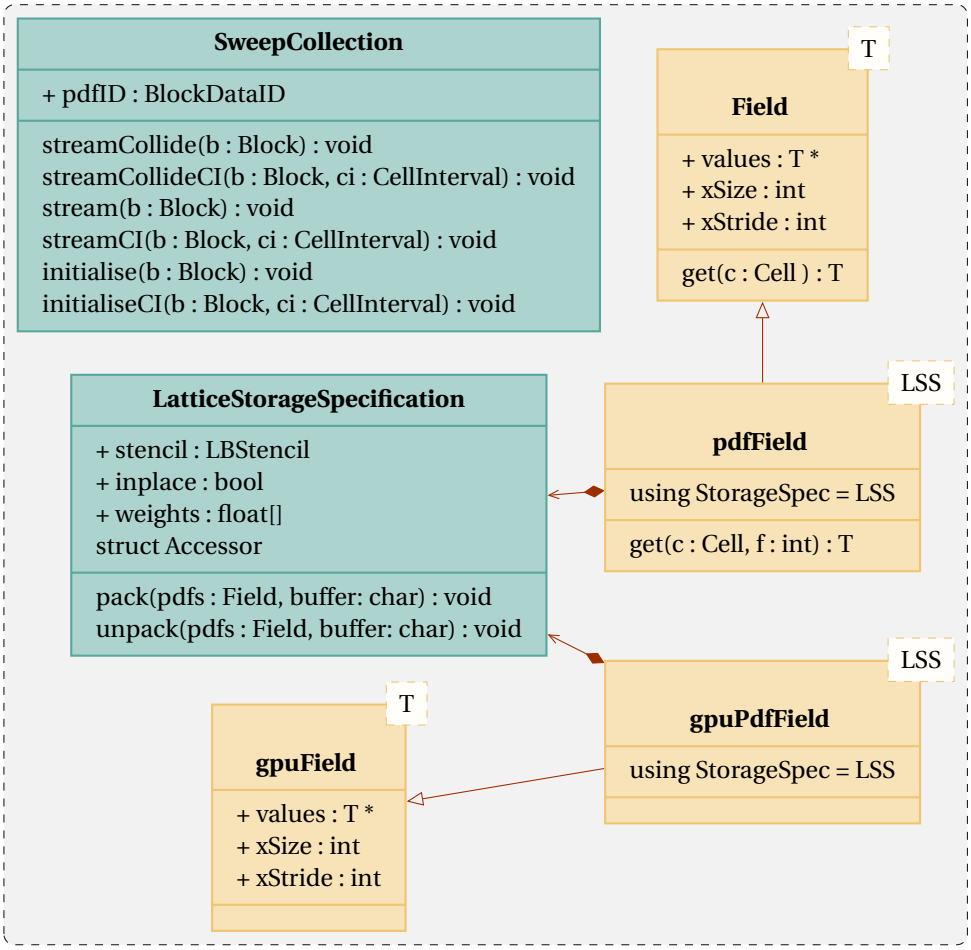


Figure 4.9: The newly developed *lbm* module of WALBERLA tightly incorporates the code generation facility with the existing framework. In this way, in most cases, static polymorphism can be avoided and replaced by specialised classes generated at compile time. The `SweepCollection` encapsulates all update sweeps that operate on WALBERLA's block structure, while the `LatticeStorageSpecification` functions as a description of the generated method. A stronger separation of the algorithmic complexity with the data structure is realised, allowing for simulations using accelerators.

The algorithmic part of the LBM is combined in a collection of `Sweeps`. In this way, `Sweeps` to execute only the streaming or collision can be obtained when needed. Additionally, all `Sweeps` come with an overloaded version receiving a `CellIntervall`, which allows the application of the operation only on parts of the data. Accessing macroscopic variables or manipulating the populations based on them is possible with respective `Sweeps`.

CHAPTER 5

MESH REFINEMENT

Contents

In this chapter, the mesh refinement algorithm of WALBERLA *will be introduced. At the start of this thesis, volumetric adaptive mesh refinement was implemented in* WALBERLA *through the work of Schornbaum [29]. Here, however, we introduce a new algorithm developed in the Bachelor thesis of Hennig [11] and supervised by the author of this work. The new algorithm leverages the code generation facilities offered by* PYSTENCILS*, providing enhanced flexibility on several levels. It is independent of the streaming pattern, and the compute kernels can be generated optimised for certain hardware, which allows the support of general purpose graphics processing units. The chapter briefly introduces mesh refinement for the lattice Boltzmann method from a general perspective in Section 5.1. Afterwards, the basic procedure of the new algorithm is shown in Section 5.2. To realise this procedure, the data exchange structure of* WALBERLA *needs strong modifications. These are shown in detail in Section 5.3. With the grid transitions, the final algorithm for the mesh refinement is presented in Section 5.4. Building on the work of Hennig [11], we implemented the new algorithm in* WALBERLA *using the newly developed integration of the code generation, which has been shown in Section 4.4.6. In this effort, the support for mesh refinement on general purpose graphics processing units was first realised, and the key elements to achieve this are shown in Section 5.5. Finally, we present a short discussion in Section 5.6 of the new developments and put them in perspective of the previous developments by Schornbaum [29].*

5.1 Grid Refinement for the Lattice Boltzmann Method

A single simulation can encompass diverse flow regimes with varying local gradients, such as laminar flow, interactions with curved boundaries, and unsteady or turbulent regions. A high spatial and temporal grid resolution is necessary to capture these variations accurately. However, in less complex regions, a coarser grid suffices. Maximising the overall grid resolution is inefficient, as it wastes computational resources on uniform flow areas. A solution to this problem is to increase the grid resolution only where needed, generally known as local mesh refinement [29, 159].

5.1.1 Parameter Scaling

One of the first contributions to show a simulation based on the lattice Boltzmann method (LBM) with one level of refinement goes back to Filippova and Hänel [159]. In their work, a two-dimensional flow around a cylinder was analysed where the region around the cylinder was resolved with a resolution $\Delta x_f = {}^{\Delta x_c}/_{n_r}$ and the resolution Δx_c was applied at the remaining domain. Their work is over twenty years old; nevertheless, some of their ideas can be considered standard nowadays in the lattice Boltzmann (LB) community. For example, the refinement factor $n_r = 2$ is commonly chosen for the LBM on refined grids [29, 60, 150, 160]. This restriction will be applied in this thesis as well. Furthermore, the refinement is not limited to the spatial resolution but extends to the temporal resolution. Thus, fixing $\Delta x_0 = \Delta t_0 = 1$ on the coarsest level for finer grid resolutions follows

$$\Delta x_k = \Delta t_k = 2^{-k} \qquad \Rightarrow \qquad c_k = \frac{\Delta x_k}{\Delta t_k} = 1, \tag{5.1}$$

where $k \in 0, 1, \ldots, N$ indicates the refinement level. Applying the refinement, both spatially and temporally, keeps the speed of sound constant in the whole simulation domain. Therefore, sound waves can travel freely between grid levels, and no reflections across the grid transitions occur. However, certain parameters need to be scaled across grid

levels to ensure the same hydrodynamic properties on all levels. Most importantly, this concerns the relaxation rate $\omega = 1/\tau$, which accounts for the kinematic viscosity of the fluid (compare Equation (3.36)). Since the kinematic viscosity must remain constant, the relaxation rate on finer grid levels must be adapted as [29]

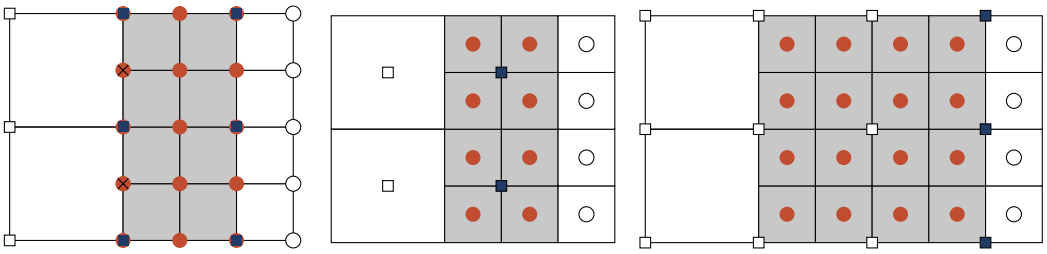
$$\omega_k = \frac{2\omega_0}{2^{k+1} + \omega_0\left(1 - 2^k\right)}. \tag{5.2}$$

5.1.2 Grid Transition Interface Layout

Numerical simulations using the LBM on nonuniform meshes can be categorised according to their grid transition interface layout [161]. A categorisation based on the work of Schukmann *et al.* [161] is presented in Section 5.1.2. In the following, these are introduced briefly.

Cell-Vertex Grid Layout

When comparing grid-level transition techniques, it is important to consider the different conceptual locations of the particle distribution function (PDF) vector on the computational grid. Due to its close link to finite difference (FD) schemes, populations are traditionally located on the vertices of the cartesian grid. This view on the data is illustrated in Figure 5.1a and referred to as cell-vertex grid layout in this work. As shown in Figure 5.1a, it leads to certain consequences at the grid transitions. Most importantly, with this grid layout, partially co-located fine nodes arise along the interface. Filippova *et al.* [159] introduced transition techniques for this type of interface layout first. Later, this technique was further developed by Lin *et al.* [162], Dupuis *et al.* [163] and Yu *et al.* [164]. One of the more prominent deficiencies of this approach arises when data from the fine grid is directly transferred to the coarse grid on co-located grid points. It is tempting to naively copy the populations as the data is located in the same location. However, it was shown that this violates the Nyquist–Shannon criterion [165] and is, therefore, not suited for turbulent simulations as it introduces spurious noise. Thus, interpolation and filtering approaches have been introduced by various authors to remove this deficiency [166–168]. Astoul *et al.* [169] made the most recent advancements of this approach. Their improvements allow for aeroacoustic simulations in the highly turbulent flow regime.



(a) Cell-vertex (b) Cell-centred (c) combined

Figure 5.1: Grid layouts for the transition between mesh levels. In the illustrations, white squares account for regular coarse nodes, while white circles account for regular fine nodes. Similarly, blue squares picture coarse interface nodes, while red circles picture fine interface nodes. Lastly, hanging nodes that only appear in (a) are marked by a black cross. All figures are inspired by the work of Schukmann *et al.* [161].

Cell-Centred Grid Layout

An alternative approach for locating the PDFs on the computational grid has been proposed by Rohde *et al.* [160]. The fundamental idea is to view the PDF array similar to a finite-volume approach. This means that a set of populations occupies the volume of the simulation cell. Hence, they are located at the cell's centre as shown in Figure 5.1b. This approach will be referred to as a cell-centre grid layout here. The major difference of this variant is that coarse and fine nodes can not share the same location. Hence, the PDFs must be interpolated in every grid direction. Additionally, the cell-vertex grid layout contains fine nodes in the middle of two coarse nodes. These nodes are called hanging nodes (marked with a black cross in Figure 5.1a). A correct treatment of these hanging nodes can be cumbersome and, thus, it can be considered an advantage if they can be avoided, as is the case in the cell-centre grid layout. Furthermore, the conservation of macroscopic quantities such as density and velocity is ensured implicitly. This grid coupling technique consists of two steps in its simplest form. The first step called explosion,

$$\boldsymbol{f}^*_{k+1}\left(\boldsymbol{x}_{k+1}, t\right) = \boldsymbol{f}^*_k\left(\boldsymbol{x}_k, t\right), \tag{5.3}$$

transfers the PDFs from one coarse cell to 2^D fine grid cells. Here, D refers to the number of spatial dimensions, and k denotes the grid level. The explosion step is illustrated in Figure 5.2 along with the second step, called coalescence,

$$\boldsymbol{f}^*_k\left(\boldsymbol{x}_k, t\right) = \frac{1}{2^D} \sum_{j=0}^{2^D} \boldsymbol{f}^*_{j,k+1}\left(\boldsymbol{x}_{k+1}, t\right). \tag{5.4}$$

The coalescence step averages 2^D fine cells and transfers the result to a single coarse cell. It can be seen as an implicit filtering step, an important feature missing in early implementations of cell-vertex transition algorithms. Filtering operations are based on the idea that the fine grid resolution contains scales that the coarse grid can not resolve. Hence, they must be eliminated [166]. Furthermore, Rohde *et al.* [160] assumed that the non-equilibrium part of the PDFs is implicitly rescaled in this method. Unlike the equilibrium part of the PDFs, the non-equilibrium part generally requires rescaling at the grid transitions, which was first discussed by Filippova *et al.* [159]. Furthermore, it must be noted that one explosion step produces two rows of valid fine cells (compare Figure 5.2). Therefore, no temporal interpolation is needed during the so-called asynchronous timestep. The asynchronous timestep refers to the second timestep executed on the fine grid at every coarse timestep due to temporal refinement. Lastly, it must be noted that the explosion algorithm in the cell-centre approach results in a lower-order interpolation. This deficiency was tackled by introducing a so-called linear explosion by Chen *et al.* [170]. Schornbaum also implemented the linear explosion in WALBERLA [29].

Combined Grid Layout

The last grid interface discussed in this thesis is a combination of the layouts above. Hence, it is referred to as a combined layout. This approach was introduced by Geier *et al.* [149], who called it compact interpolation. The idea is to locate the coarse grid nodes similar to the cell-vertex approach and the fine nodes similar to the cell-center approach, as illustrated in Figure 5.1c. The resulting algorithm uses an overlapping region of two coarse cells. One of the most fundamental ideas of the combined grid layout is to analyse the

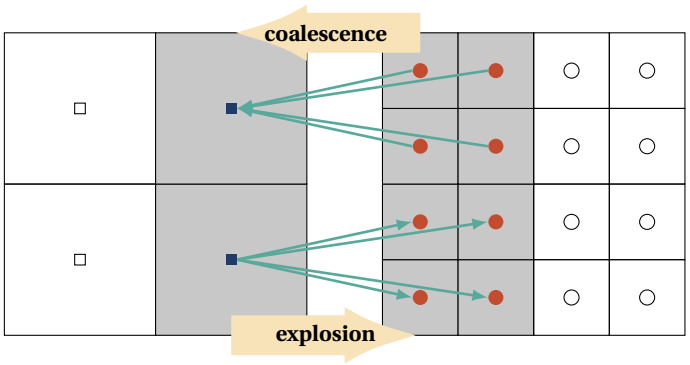


Figure 5.2: Grid transition for cell-centred grid layout in two dimensions. An overlapping region of one coarse cell and two fine cells is used at the interface. An explosion step (compare Equation (5.3)) is performed to transfer the PDFs from the coarse to the fine grid points. In the other direction, a coalescence on the fine grid nodes is performed (compare Equation (5.4)). In the illustration, white squares account for regular coarse nodes, while white circles account for regular fine nodes. Similarly, blue squares picture coarse interface nodes, while red circles picture fine interface nodes.

macroscopic equation that the LBM should recover. In the case of the hydrodynamic LBM, these are the Navier-Stokes equations (NSEs). Since the NSEs contain second-order derivatives of the momentum field, the grid transition should also be second-order accurate. In Geier's approach, this is done by calculating the macroscopic density and velocity. The velocity on the next finer grid nodes is then interpolated quadratically and the density linearly (due to the first order pressure gradient in the NSEs). The newly calculated macroscopic values are then used to calculate the equilibrium part of the PDFs on the fine nodes. Lastly, the non-equilibrium part is rescaled using similar ideas, and the PDFs can be recovered correctly [149, 171]. The naming compact interpolation comes from the fact that the order of interpolation achieved by Geier *et al.* [149] is usually not achievable using only a first neighbourhood. This is only possible by employing the local information encoded in the moments of the PDFs. Hence, this can be seen as a clear advantage of the method. Nevertheless, it must be emphasised that the overlapping region is larger than in the cell-center approach.

5.2 Basic Procedure for Volumetric Grid Refinement

The refinement procedure covered in this thesis is based on volumetric grid refinement and thus relies on the cell-centred grid layout, originally developed by Rohde *et al.* [160]. Later, the original algorithm was implemented in WALBERLA by Schornbaum [29], where the focus was to adapt the algorithm for massively parallel environments. The algorithm developed by Hennig [11] builds on the work of Schornbaum and introduces certain modifications. The most significant modification is that Schornbaum partly separated the stream-collide algorithm. This means only the streaming was executed at certain steps, and likewise, only the collision on other steps. As this causes more passes through the PDF array, it can lead to a performance overhead [172].

The adapted procedure is illustrated in Figure 5.3. For the sake of simplicity, only the stationary and two additional populations are shown. As mentioned before, most refinement algorithms for the LBM rely on temporal refinement in addition to spatial refinement. For this reason, an additional asynchronous timestep must be executed on the fine grid. This is indicated by timestep $t_{1/2}$ on the timeline in Figure 5.3. The procedure begins at the pre-collision state of the PDFs. The pre-collision state is either set during the initialisation phase of a simulation or reached at the end of one timestep on the coarsest grid. At t_0, the stream collide step can be executed on all grid levels, which yields the post-collision state indicated as state A in Figure 5.3.

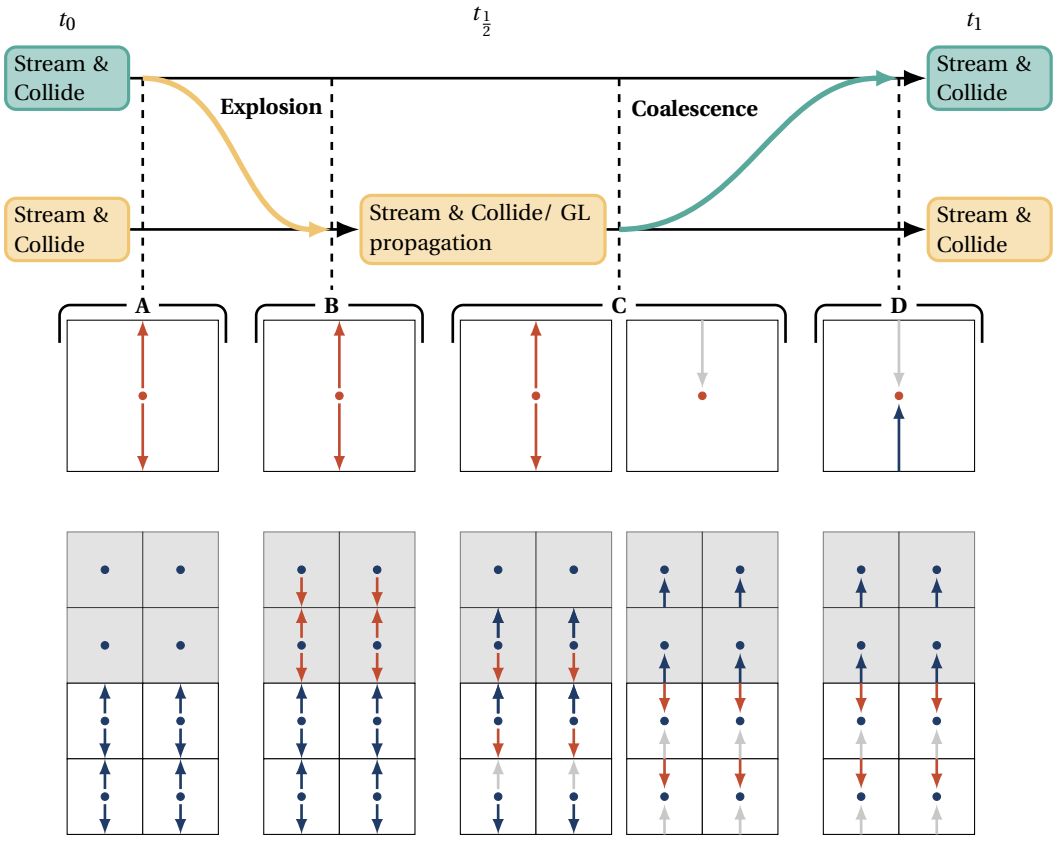


Figure 5.3: Time stepping procedure for the LBM using two grid levels. For the sake of simplicity, only the stationary and two additional populations are shown. The procedure relies on four steps, labelled A, B, C and D. At the top, a timeline is shown as discrete timesteps. The steps of the coarse grid are labelled in green, while the steps on the fine grid are labelled in yellow. One coarse interface cell and four corresponding fine cells are shown for each state. The fine interface cells are marked in grey. While the illustration is shown in a two-dimensional setting, it must be emphasised that the procedure also works for common three-dimensional lattice stencils like the D3Q19 or D3Q27 stencil.

At this stage, the information in the ghost layers is no longer valid. Therefore, a uniform redistribution (explosion) according to Equation (5.3) needs to be performed. Consequently, the two ghost layers of the fine gird contain valid populations in state B. This allows the execution of a stream-collide step on the fine cells, leading to state C. It is im-

portant to note that the ghost layer populations also need to propagate. Thus, a streaming step is executed between states B and C on the first ghost layer of the fine grid cells. This is the only part of the adapted algorithm where a streaming step is executed without a collision.

The post-collision populations on the fine grid that stream into the coarse grid are now shifted to the respective coarse location according to Equation (5.4). This produces state D, a pre-collision state for all PDFs on all gird levels. Therefore, the procedure is completed, and the next timestep can start.

Besides the more complex time-stepping structure and the redistribution and coalescence step, all ingredients needed in the procedure shown in Figure 5.3 are also needed for simulations on uniform domains. In the work of Schornbaum, these steps are directly integrated into the communication routines of WALBERLA [29]. Similarly, this idea is followed by Hennig and will be explained in the following sections.

5.3 Extension of the Data Exchange Structure

The communication infrastructure of WALBERLA was introduced in Section 4.4.4. In this section, it is extended by the redistribution and coalescence routines. As detailed before, the communication infrastructure can be divided into three logical parts, all of which must be extended when the grid is non-uniform. The first part is the communication scheme. The communication scheme can be viewed as the manager of the data exchange between WALBERLA's blocks. Thus, it traverses the octree data structure and checks each block for data exchanges that need to be performed with its neighbouring blocks. In Section 4.4, the data exchange concerned only blocks with the same spatial resolution. With non-uniform grids, the communication scheme needs to consider the resolution of its neighbours to call the correct procedures respectively. Due to the modular design of WALBERLA, the communication scheme implemented by Schornbaum (called `NonUniformBufferedScheme`) can be used directly with the modification shown in this thesis without changes.

On the other side, the pack info needs to call the compute kernels that perform the data exchange. Since these compute kernels are fundamentally changed, the pack info needs to be adapted. A simplified version of its interface is shown in listing 5.1. The interface is still similar to the work of Schornbaum, which is important to keep it compatible with the communication scheme. However, the implementation details differ strongly. Like its uniform counterpart, the non-uniform pack info eventually performs calls to specialised compute kernels generated by PYSTENCILS. These specialised compute kernels are integrated into the generated `LatticeStorageSpecification`, specifically for a particular platform, which allows their execution on general purpose graphics processing units (GPGPUs) as well. Second, a specialised version of the various streaming patterns for each lattice direction is generated.

In the following sections, the implementation details of the coarse-to-fine and the fine-to-coarse communication will be presented.

5.3.1 Coarse-to-Fine Communication

The data exchange between a coarse block and its adjacent fine blocks follows well-defined rules. For a coarse block, every population streaming out of a cell on the interface layer must be packed into a communication buffer. These cells are redistributed to the respective ghost layers on the adjacent fine blocks. Here, it must be noted that all cells

CODE LISTING 5.1: Simplified interface for non-uniform pack info class. The subclasses must implement the interface methods.

```
class NonUniformPackInfo {
   /* Equal level communication */
   void packDataEQ(Block * sender, stencilDirection dir, SendBuffer & buffer);
   void unpackDataEQ(Block * receiver, stencilDirection dir, RecvBuffer & buffer);
   void communicateLocalEQ(Block * sender, Block * receiver, stencilDirection dir);

   /* Coarse to Fine communication */
   void packDataCF(Block * sender, stencilDirection dir, SendBuffer & buffer);
   void unpackDataCF(Block * receiver, stencilDirection dir, RecvBuffer & buffer);
   void communicateLocalCF(Block * sender, Block * receiver, stencilDirection dir);

   /* Fine to Coarse communication */
   void packDataFC(Block * sender, stencilDirection dir, SendBuffer & buffer);
   void unpackDataFC(Block * receiver, stencilDirection dir, RecvBuffer & buffer);
   void communicateLocalFC(Block * sender, Block * receiver, stencilDirection dir);
};
```

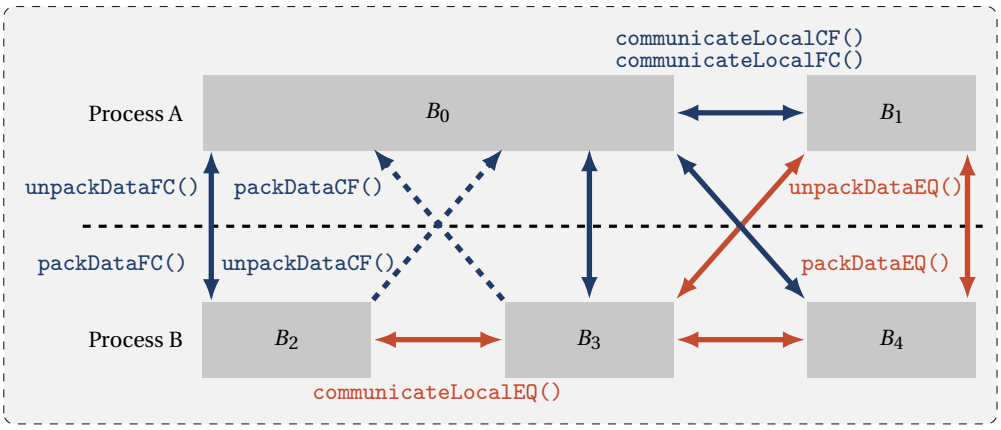
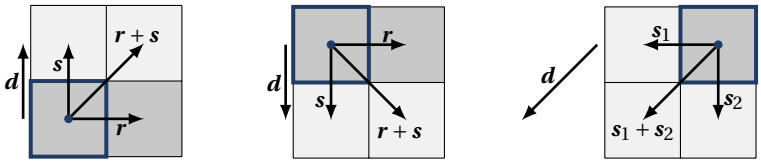


Figure 5.4: The data exchange within WALBERLA's block structure. The resolution of the blocks indicates their size. Thus, block B_0 occupies space on mesh level k, while all other blocks resemble parts of the grid on level $k+1$. Red arrows show communications between blocks of the same resolution, while blue arrows indicate communication passes between blocks of different resolutions. The respective function call to the pack info (compare listing 5.1) is shown for some arrows. Dashed lines highlight asymmetric links. For these links, data is only communicated from the fine blocks to the coarse block.

are redistributed only on the first ghost layer. In contrast, only the missing populations for the following ghost layer propagation are written on the second ghost layer. If all blocks reside in the same process, this procedure is realised directly on the respective data pointers. Thus, the communication buffer is not needed in these cases. In coarse-to-fine communication, the packing of the data is relatively straightforward since the complete population set must be packed on the corresponding coarse cells. However, unpacking the data is more complicated as every coarse cell corresponds to several fine cells on which only specific populations are written. Therefore, it will be explained in more detail in the following sections.

Unpacking and Redistribution

For the unpacking and redistribution routine, the first ghost layer of the PDF array of the fine blocks is iterated with a step size of two. A set of 2^D cells must be written in each iteration pass. However, only specific populations must be updated in each of these cells. This information is extracted from the lattice direction $\boldsymbol{d}$, an input parameter of the unpacking routine (compare listing 5.1). The lattice direction $\boldsymbol{d}$ defines the data's origin; therefore, the same rules can be applied depending on this direction. Similar to the equal-level communication routines, a specialised compute kernel is generated with PYSTENCILS for each direction. An illustration for three exemplary directions is pictured in Figure 5.5.



(a) Northern slice (b) Southern slice (c) South-western slice

Figure 5.5: Illustration of the first ghost layer root cell and the offset to the first and second ghost layer. The code generator computes these offsets during the code compilation. An arrow on the left indicates the direction of the coarse source block. The first ghost layer is highlighted in dark grey, and a blue frame indicates the root cell. Principal orthogonal vectors are denoted as $\boldsymbol{r}$, and the principal subdirections are denoted as $\boldsymbol{s}$.

First, in the code generation phase, a set of principal directions $\boldsymbol{e}_\alpha$ is computed, i.e., canonical basis vectors of $\mathbb{R}^D$. The principal directions are orthogonal to $\boldsymbol{d}$ and the oriented principal directions $\pm\boldsymbol{e}_\alpha$ that are contained in $\boldsymbol{d}$. These are denoted as principal subdirections. The cell group fills a cubic volume within the ghost layers, and within the cube, every cell is identified by an offset $\boldsymbol{r} \in \{0,1\}^D$. The *root cell* of each cell group is identified by the smallest offset $\boldsymbol{r}$. This makes it possible to identify for each cell if it lies in the first or the second ghost layer.

The first ghost layer extends into all principal directions orthogonal to the communication direction. Hence, all of its cells in the group can be found by adding all linear combinations (with coefficients $\in \{0,1\}$) of the orthogonal vectors to the root cell location. The cells of the second ghost layer are reached by adding any linear combination of the principal subdirections of $\boldsymbol{d}$ onto the first ghost layer vectors. In Figure 5.5, three exemplary cases show the cell offsets as a combination of the orthogonal and subdirections. Finally, all cell offsets are identified, and the copy assignments can be generated by PYSTENCILS. It must be emphasised that while the described procedure seems complicated, it is entirely resolved during the code generation phase. The generated compute kernels then only contain a regular loop over the first ghost layer of the fine cells and the store instructions for each loop iteration.

As shown in listing 5.1, all functions of the pack info have a very similar interface. The packing and unpacking functions receive pointers to the sender and receiver blocks, respectively, and a pointer to the buffer array and the stencil direction $\boldsymbol{d}$. However, their implementation differs vastly. The reason for this is illustrated in Figure 5.6. Data exchange between blocks on the same grid level is more straightforward because the ghost layer of one block only overlaps with one neighbouring block, and the amount of data that needs to be sent to a neighbour is the same as the amount of data sent from this block.

Thus, the communication is always symmetric. In the coarse-to-fine communication, this is no longer the case. As one coarse block can be adjacent to multiple fine blocks at a direction $\boldsymbol{d}$, some cells overlap with multiple blocks. Hence, the pack info needs to extract the corresponding pointers to the data arrays and the corresponding cell intervals that need to be communicated. An example is an edge intersection as pictured in Figure 5.6.

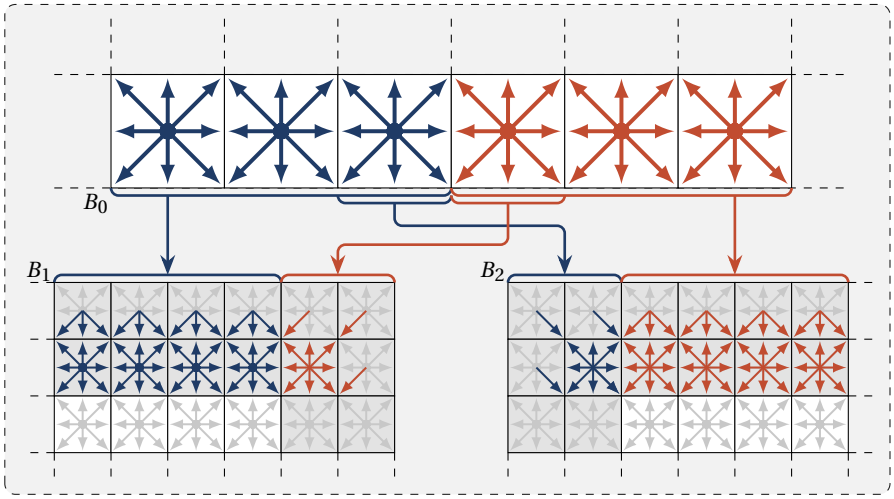


Figure 5.6: Illustration of the coarse to fine communication across a two-dimensional edge. The coarse block B_0 shadows cells on two fine blocks B_1 and B_2. Hence, different subsets of the population vector must be unpacked at each fine block.

Since communication can be asymmetric, this needs to be respected in the packing and unpacking routines. Through WALBERLA's octree, the relative location of neighbouring blocks can be inferred. This is possible through the strict setup of the block identifier, as explained in detail by Schornbaum [29]. From the identifier, the orthogonal offset vector $\boldsymbol{r} \in \{-1,0,1\}^D$ can be constructed, with nonzero entries in every direction orthogonal to the interface. A negative value denotes a fine block location at the low end of the coarse block edge or face, while a positive value denotes a fine block location at the high end of the coarse block edge or face. In this manner, the locations of block B_1 and B_2 with respect to block B_0 can be identified precisely. The correct cell intervals are extracted for each of these blocks, and the correct unpack kernels can be called.

5.3.2 Fine-to-Coarse Communication

The fine-to-coarse communication propagates populations from one or more fine blocks to a coarse block as described by Equation (5.4). Implementation-wise, the biggest challenge is that one coarse cell can be represented by a group of fine cells due to WALBERLA's data structure, which expresses overlapping regions exclusively via the ghost layers around the blocks. This group of fine cells $\boldsymbol{x}_j^k$ can be seen as instances of a virtual cell $\boldsymbol{x}_j$, representing the data exchange. The problem is that this cell group is not always synchronised because there is no reason to force these cells to be synchronised. Hence, their global state may differ. During the fine-to-coarse communication, only the latest version of each population in each of these cell groups is allowed to participate in the coalescence step. However, it is not trivial to find these populations a priori.

The described problem leads to a reformulation of Equation (5.4) in a distributed form. It reads

$$\boldsymbol{p}^i(\boldsymbol{x}_k, t) = \sum_{j \in J^i} \boldsymbol{f}^*_{j,k+1}(\boldsymbol{x}_{k+1}, t), \tag{5.5}$$

$$\boldsymbol{f}^*_k(\boldsymbol{x}_k, t) = \frac{1}{2^D} \sum_{i=1}^{n} \boldsymbol{p}^i(\boldsymbol{x}_k, t), \tag{5.6}$$

where $\boldsymbol{p}^i$ describes a partial summation for each fine fine block B_i. The summation index $J^i \subseteq \{0, \ldots, 2^D - 1\}$ represents the subset of populations considered in the summation. The partial summations are transferred to the coarse block, and the coalescence is completed.

Origin Cell Determination

Forming the partial summation in Equation (5.5) is a complex task as it requires determining which cells and populations must be included. It must be ensured that every population (and its virtual versions) is only considered once. Furthermore, this information must be inquired locally for each block to avoid jeopardising the simulation's performance. The populations of interest can be found by following the steps shown in Figure 5.3 backwards to an origin cell $\boldsymbol{o}$. For this cell, criteria must be established to select the correct populations.

The definition of the origin cell $\boldsymbol{o}$ is simple. It must be the cell where a population was modified most recently. From the perspective of a fine block, two possibilities arise. Either a population that streams into the coalescence region was propagated before on a ghost layer that shadows a coarse block or not. If the cell was altered during such a propagation step, its distance is two lattice cells. Otherwise, it is a direct neighbour of the cell in the coalescence region. Now that the originating cell is determined, the populations considered for the partial sum must be checked. For this, the following cases exist.

Fine Grid Collision If the population streams into the coalescence region from the block's interior region, it must be included. If it streams into the coalescence region from the ghost layer, it is shadowed by another fine block, and this fine block needs to be taken into account.

Corner skipping For better understanding corner skipping is illustrated in Figure 5.7. Essentially, this is the case at concave corners or edges where two coarse blocks could communicate directly with each other (because they are neighbours). However, the communication must be performed in two steps via the finer block, which is adjacent. The reason is that a population is split on the fine block during the explosion step. Some of the split populations will enter the interior region of the fine block, and some will enter the next coarse block. This case may include several fine blocks, all holding synchronous populations. In this case, the block with the smallest identifier is chosen.

Boundary handling If the origin cell is a boundary cell, the boundary handling will replace either the collision operation or the uniform redistribution on $\boldsymbol{o}$. This happens automatically by applying the boundary routines. Therefore, the population that needs to be considered for coalescence can be determined using the same rules described before.

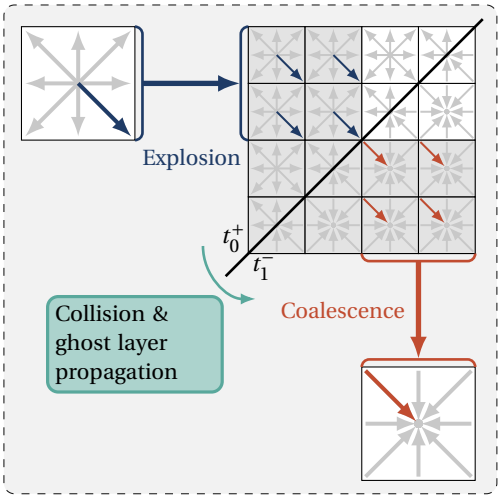


Figure 5.7: Illustration of coarse-fine-coarse corner skipping. First, a population is redistributed from a coarse block to a fine block. On the fine block, the population travels one cell during the ghost layer propagation, and afterwards, it enters the coalescence step without participating in any collision step. It looks like the population could be communicated directly between the coarse blocks. However, this does not work since some of its redistributed versions do enter collision steps on the fine block. Hence, the equal level of communication between the coarse blocks must be skipped.

Realisation using Bit Masks

Now that the criteria for the populations have been defined to determine whether they participate in the coalescence step, they need to be executed during the runtime. A naive solution would be to introduce branches that check for these criteria for each population. However, branches can lead to complex memory access patterns and eventually prohibit essential optimisations in the compilation phase. This situation is worse on GPGPUs as threads are organised in groups, and the threads of a group can only execute the same operation. Thus, excessive branching in a compute kernel might lead to idle threads waiting for other threads to execute their operation. This problem is called warp divergence.

An idea to avoid any branching is to use a bitmask. A bit mask is an additional data array with an integer value for each cell. Here, a 32-bit integer is used, which is enough to represent the 27 populations that reside in each cell on the largest lattice stencil supported by WALBERLA. For each of the origin cells $\boldsymbol{o}$ determined through the strategy above, a bit is set for each population considered in the coalescence step. Then, the partial summation can be rewritten as:

$$\boldsymbol{p}^i(\boldsymbol{x}_k, t) = \frac{1}{2^D} \sum_{j=0}^{2^D} \boldsymbol{b}_j(\boldsymbol{x}_{k+1}, t)\, \boldsymbol{f}^*_{j,k+1}(\boldsymbol{x}_{k+1}, t)\,, \tag{5.7}$$

where the bitmask is represented by $\boldsymbol{b}$. This equation can be realised in the resulting compute kernel as shown in listing 5.2, where the bit operation checks if a certain bit is set on the i^{th} location. If the bit is set, the operation returns 1, and the population is respected in the summation. Otherwise, the operation returns 0, which excludes the population in the summation. These bitwise operations can be generated with PYSTENCILS using the function `flag_cond`. This function derives from *SymPy*'s abstract syntax tree (AST) structure and can, thus, be integrated into the symbolic expressions that enter PYSTEN-

CILS's code generation. The function `flag_cond` hast three input arguments. The first one represents the symbolic bit mask which will be used for the flag evaluation in the generated kernel. The other two arguments are the bit location i of the bit that should be evaluated, and the last argument is an expression that will be executed if a condition is true. This expression is a `Field.Access` to the symbolic PDF array in the case discussed here. Hence, listing 5.2 can be fully described symbolically using PYSTENCILS.

CODE LISTING 5.2: Exemplary implementation of bit-masked partial coalescence. The qualification multiplier is computed through bitwise operations on the bit mask. The listing shows the summation of a group of four cells on a two-dimensional lattice.

```
p[i] = ((b[x  , y  , z] >> i) & 1) * f[x  , y  , z, i]
     + ((b[x+1, y  , z] >> i) & 1) * f[x+1, y  , z, i]
     + ((b[x  , y+1, z] >> i) & 1) * f[x  , y+1, z, i]
     + ((b[x+1, y+1, z] >> i) & 1) * f[x+1, y+1, z, i];
```

Implementation Remarks

The main ideas behind the coalescence are established now, and the final implementation of the pack info is the last missing piece. In this section, some important remarks on integration need to be made. First, the bitmask is handled inside the pack info and set up during the initial phase. As long as the domain partition is constant, the bitmask is constant, too. In the setup of the bitmask, every block with a neighbouring block of higher resolution is iterated, and the criteria discussed before are encoded in the bitmask. Hence, it requires only a single iteration through the local blocks of each process, and it can be executed fully parallel. The bitmask must be reinitialised if the block structure changes during a simulation run, e.g., during adaptive grid refinement. Another important point concerns Equation (5.6). The partial sums $\boldsymbol{p}$ that need to be summed up to update the corresponding populations on the coarse grid can come from different locations. Hence, these are applied successively to the destination population, and it is necessary to set them to zero to ensure correctness in a parallel environment. This is done using the function `zeroCoalescenceRegion` before the summation is executed. The function `zeroCoalescenceRegion` is also generated by PYSTENCILS and is executed before the coarse-fine-communication.

5.4 Implementation of the Recursive Time Step

After a detailed view of the grid transition between blocks of different resolutions, all that remains is to discuss the final time-stepping algorithm that arises, shown in Algorithm 4. Since the mesh is refined spatially and temporally, creating a recursive function that realises the descent to finer grids with binary recursion is natural. This results in executing 2^k timesteps on grid level k. The algorithm starts with the combined stream collide function on every grid level. Afterwards, the data exchange between the blocks happens. For this, the order must be followed strictly. First, coarse-to-fine communication is executed, followed by equal-level communication, boundary handling, and fine-to-coarse communication. The order is important because only certain populations are in a valid state. For example, the coalescence is only correct after the boundary handling updates the origin cells marked with a boundary flag.

The subcycle, or asynchronous step, starts with the ghost layer propagation to ensure the correct population in the first ghost layer of the fine grid. This allows the execution of the combined stream collide algorithm followed by a sequence of communication functions with the boundary handling. Again, the order is important here.

Algorithm 4: The recursive time stepping scheme. The current grid level is indicated k and N represents the globally finest grid level. Every function takes the grid level as an argument to indicate on which blocks their respective operations are executed.

```
1  function recursiveTimestep(k, N):
2      streamCollide(k)
3      if k < N then
4          recursiveTimestep(k+1, N)
5      end
6      if k ≠ 0 then
7          coarseToFineCommunication(k−1)
8      end
9      equalLevelCommunication(k)
10     boundaryHandling(k)
11     if k ≠ N then
12         fineToCoarseCommunication(k+1)
13     end
14     if k == 0 then
15         return
16     end
17
18     ghostLayerPropagation(k)
19     streamCollide(k)
20     if k ≠ N then
21         recursiveTimestep(k+1, N)
22     end
23     equalLevelCommunication(k)
24     boundaryHandling(k)
25     if k ≠ N then
26         fineToCoarseCommunication(k+1)
27     end
28 end
```

5.5 Extension for GPGPU Architectures

In contrast to the work of Schornbaum [29], the compute kernels of the refinement algorithm developed by Hennig are automatically generated by PYSTENCILS. Therefore, they can be explicitly generated for every target hardware supported by PYSTENCILS. This includes accelerators like GPGPUs. At the start of this thesis, WALBERLA had no support for accelerators on refined grids. Hence, no infrastructure existed to support these scenarios. In the following, the most important extensions of WALBERLA will be explained, which have been realised through this thesis to include the generated compute kernels for accelerators.

For the central processing unit (CPU) version of the refinement algorithm, the communication scheme was already provided by Schornbaum, as mentioned before. Since this work was built upon the same application programming interface (API), it was possible to use his communication scheme directly. To support GPGPUs the first task was to provide such a communication scheme. The newly developed `NonUniformGPUScheme` has the same role as its counterpart for CPUs. Essentially, it traverses the octree of WALBERLA and calls the respective communication function of the pack info class to realise the data exchange between blocks.

A significant difference, among others, is that the communication buffers now point to device memory. Hence, an additional structure needs to be realised to manage these. Unlike their CPU counterparts, it is unclear if these buffers can be communicated directly between message passing interface (MPI) ranks. This depends on the MPI library used. If pointers to device memory are not understood by the MPI library, the buffers must be synchronised to the host memory and communicated from there. On the receiving rank, they need to be synchronised to the respective device and unpacked from the buffer array.

Furthermore, the non-uniform pack info class is also ported for GPGPU support, where the essential difference is that the class needs to extract data pointers from the `gpuPdfField`. Additionally, the bit mask used in the coalescence must be synchronised to the device memory.

5.6 Discussion

The method presented for volumetric mesh refinement for the LBM in WALBERLA shows several significant advantages over the previous developments by Schornbaum [29]. The most important advantage is the lowered memory footprint. As shown in the previous sections, the overlapping regions through the ghost layers around the blocks impose high complexities on the implementation side. Schornbaum solved this problem with four ghost layers around each block. Even though not all ghost layers are involved in communication steps, they still leave a higher memory footprint on the simulation overall. Especially when the individual blocks are small, the ghost layers quickly impose a significant overhead. On top of that, the implementation by Schornbaum was strictly tailored to the pull streaming pattern, which means it relied on two PDF arrays. To illustrate this problem more clearly, Figure 5.8 shows the memory consumption of cubic blocks compared between Schornbaum's implementation and the one presented here. This means we divide the memory consumption of the implementation shown here by the one of Schornbaum. The memory consumption M_f is computed by

$$M_f = (n + n_{\mathrm{gl}})^3 n_{\mathrm{b}}, \tag{5.8}$$

where n represents the side length of a block in lattice cells, and n_{gl} is the number of ghost layers around the block. Furthermore, the number of bytes per cell n_{b} is calculated by multiplying the number of values per cell by eight (for double precision). In the case of a D3Q19 stencil, the number of values per cell is 19 for in-place streaming patterns and 38 for two field approaches. As shown in Figure 5.8, the overhead is especially large for smaller block sizes. In the cases of 8^3 blocks, the new developments need only about one-fifth of the memory compared to the old approach. The saving becomes less significant for larger blocks, but due to the in-place streaming pattern, it will always be less than half.

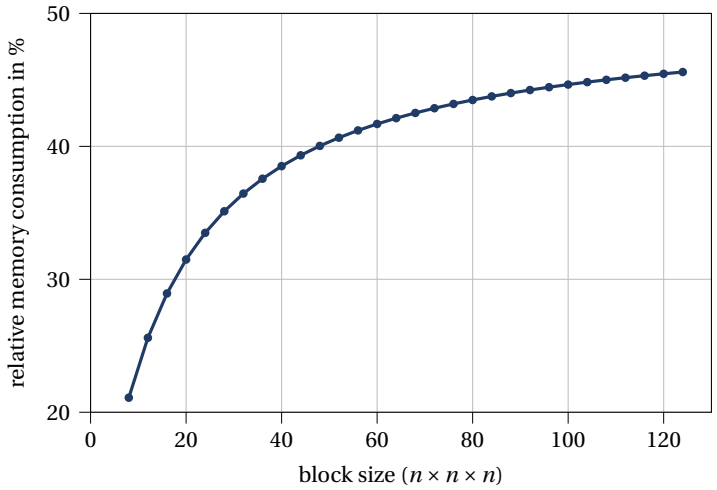


Figure 5.8: Relative memory consumption per cubic block size. Comparison between present work and Schornbaum [29]. Due to the usage of only two ghost layers and in-place streaming, the present approach only needs between 21 % and 46 % of the memory compared to Schornbaum.

Besides the advancements that have been realised throughout the scope of this thesis, some remaining problems should be discussed here.

Further memory reduction The memory reduction that was realised through this thesis is significant. Nevertheless, especially when smaller block sizes are used ($\leq 16^3$) the ghost layers around the blocks are still a dominating factor in the overall memory consumption. The problem is that the ghost layers are always fixed around every block. However, two ghost layers are only needed at grid transitions, and blocks of equal size on the same process would not need any ghost layers at all. Furthermore, buffer arrays could be used directly to replace the ghost layers. As shown in Figure 4.6 most of the values in the ghost layers are not needed at all.

Overlapping region At the time of writing, overlapping regions in WALBERLA are exclusively realised through the ghost layers. The emerging grid transition is rather complex because cells can be shadowed in multiple locations. This problem could be solved by changing the topology of the block structure and allowing for actual overlapping regions.

Interpolation accuracy The interpolation algorithm realised in this work is of a lower order and multiple times, it was shown that at least a fourth-order interpolation method would be desirable at the grid transition for turbulent flows [169, 171].

CHAPTER 6

TURBULENT SINGLE-PHASE FLOWS

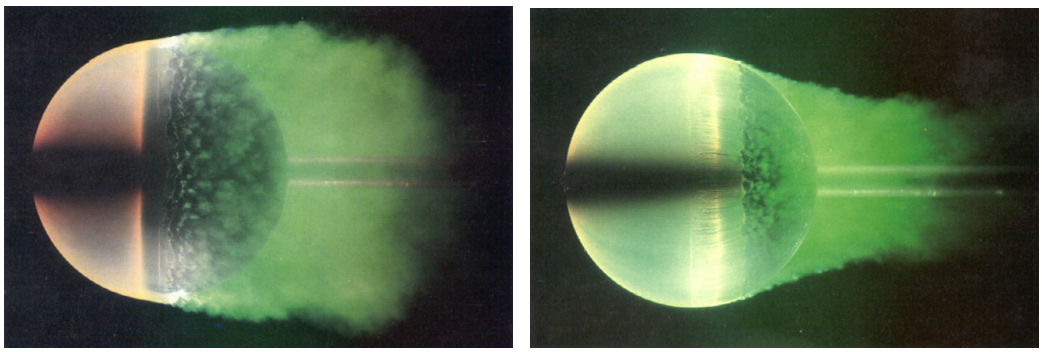
Contents

In this section, we demonstrate the capabilities of WALBERLA *to predict the drag crisis of a spherical object. This phenomenon is intrinsically complex because it occurs in a highly turbulent flow regime. Therefore, it requires a capable flow solver, suitable boundary conditions, and extremely fine mesh resolution. Static mesh refinement is used around the sphere to provide the needed mesh resolution while keeping the computational cost acceptable. First, we briefly introduce this phenomenon in Section 6.1. Afterwards, we introduce the simulation setup in detail in Section 6.2, followed by the numerical results in Section 6.3. Finally, we present our concluding thoughts in Section 6.4.*

6.1 Brief Introduction

A pivoting parameter in computational fluid dynamics (CFD) is the drag coefficient c_D. It quantifies the flow resistance of an object in the flow field [18]. Hence, it is unsurprising that this parameter was carefully investigated as soon as engineers started conducting wind tunnel experiments. One of the earliest rigorous studies of flow phenomena in wind tunnels goes back to the work of Gustav Eiffel [173] at the beginning of the 20th century. After constructing the world's first tower with a height of more than 300 meters, he built a wind tunnel at the foot of this tower [174]. In this wind tunnel, he carefully measured the drag coefficient of different bodies. To his surprise, he realised that the drag coefficient of a spherical body shows a sudden drop at Reynolds numbers around Re = 250 000. This sudden drop in the drag coefficient is now known as the drag crisis, and due to one of its first investigators, it is sometimes referred to as the Eiffel paradox.

Water tunnel experiments of the flow around a spherical object have been conducted in the past by Office national d'études et de recherches aérospatiales (ONERA) [1]. The resulting flow field is shown Figure 6.1 [174]. It clearly shows the different flow behaviour when increasing the Reynolds number from Re = 200 000 to Re = 300 000. At high Reynolds numbers, the flow field shows less perturbations.



(a) Re = 200 000 (b) Re = 300 000

Figure 6.1: Experimental investigation of the drag crisis in a water tunnel by ONERA. The experiment shows the flow field in (a) at the sub-critical region at a Reynolds number of Re = 200 000 and in (b) at the super-critical region at a Reynolds number of Re = 300 000. The images are taken from [174].

[1] https://www.onera.fr/en

More than a hundred years after Eiffel's experiments, many wind tunnels have been transferred to the digital world, and computers predict the flow field by solving numerical equations. While this is an enormous development, engineers nowadays must ensure that their numerical experiments can capture complex phenomena like the drag crisis precisely. At the drag crisis, dissipation drops by around 80 % [175]. This drop in energy dissipation caused by the smaller eddies influences engineering design decisions significantly and thus must be reproduced correctly. As explained in the work of Geier *et al.*, a numerical method shows a misprediction of the dissipation by around 400 % if it fails to reproduce the drag crisis [60]. Thus, it must be considered qualitatively wrong.

This thesis chapter investigates the flow around a sphere under highly turbulent conditions. On the one hand, these simulations aim to validate the implemented mesh refinement approach. So far, the mesh refinement capabilities of waLBerla have been studied exclusively on central processing units (CPUs) [29, 176, 177]. On the other hand, the simulations also prove the suitability of LBMpy's collision operators and boundary conditions. In this chapter, all experiments are conducted on LUMI-G [2] and these are the first simulations of non-uniform meshes on general purpose graphics processing units (GPGPUs) with waLBerla. As shown in Chapter 5, the algorithm for refined grids has been entirely re-implemented by Hennig [11] and implemented in waLBerla as part of this thesis. While validations in standard academic test cases have been performed as part of this effort, complex flow phenomena under turbulent conditions have not been investigated yet. Besides the ability to refine the computational grid at regions of interest, this flow problem additionally asks for a suitable collision model for the lattice Boltzmann method (LBM) and boundary conditions to match the geometry [60].

6.2 Simulation Setup

The setup for simulating the flow around a spherical object in a wind tunnel appears in Figure 6.2. The simulation domain consists of a cubic channel sized $2D \times D \times D$, where $D = 20d$ and d represents the sphere's diameter. The sphere's centre is placed $12d$ from the western side of the box. At $x = 0$, velocity bounce-back (UBB) boundary conditions introduce an inlet velocity of $u_0 = 0.05^{\Delta x_0/\Delta t_0}$. To manage the outflow at the eastern side, a sponge layer of length $4d$ combines with an anti-bounce-back boundary condition. This sponge layer smoothly reduces the coarse grid's relaxation rate ω to 1.9 at the outlet by applying a hyperbolic tangent, ensuring stability in the outflow boundary condition and minimising pressure reflections. Free slip boundary conditions are applied to all other sides, while the sphere uses interpolated bounce-back boundary conditions according to Equation (2.28). The coarsest grid level has a resolution of $\Delta x_0 = 1/16$, which translates to $\Delta x_6 = 1/512$ at the finest level. Thus, the sphere's diameter is resolved with 512 lattice cells. The coarsest grid level has a timestep size of $\Delta t_0 = 1/320$, and the simulation runs for 60 000 coarse timesteps, leading to nearly two million timesteps on the finest grid level. The setup uses 3726 blocks, each containing 64^3 cells, resulting in a total of $9.76 \cdot 10^8$ simulation cells across all levels. The simulations shown here have been performed in under ten hours using 32 AMD MI250X GPGPUs. The K17 cumulant collision operator, with a limiter of 0.01, handles the collision step of the LBM. In this experiment, the Reynolds number is

[2] https://docs.lumi-supercomputer.eu/hardware/lumig/

defined using the sphere's diameter as the reference length scale,

$$\mathrm{Re} = \frac{u_0 d}{\nu_0}. \tag{6.1}$$

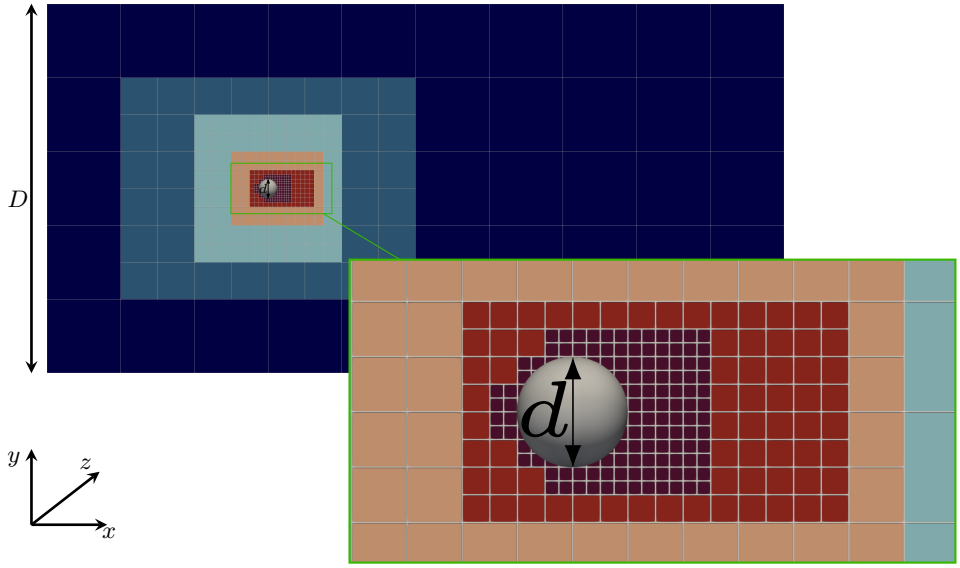


Figure 6.2: Simulation domain to investigate the flow field of a spherical object in a wind tunnel. The setup shows the xy-plane at the centre of the sphere. Six layers of exponentially refined grids have been used to resolve the sphere's diameter with 512 grid cells. The domain size is $2D \times D \times D$ with $D = 20d$ and d is the sphere's diameter.

6.3 Simulation Results

The drag coefficient c_D for a spherical body is defined as:

$$c_\mathrm{D} = \frac{8F_\mathrm{D}}{\rho u_0 \pi d^2}, \tag{6.2}$$

where ρ refers to the mean density of the fluid. This work considers the density fluctuations negligible; thus, the mean density is assumed to be unity. Hence, the only missing parameter in Equation (6.2) is the force on the sphere F_D. A standard way to calculate the fluid's force on a body is to apply the momentum exchange method [178]. The momentum exchange between two opposing directions of neighbouring cells is

$$\boldsymbol{c}_i f_i(t, \boldsymbol{x}) - \boldsymbol{c}_{\bar{i}} f_{\bar{i}}(t, \boldsymbol{x} + \boldsymbol{c}_i \Delta x). \tag{6.3}$$

With this equation, the momentum exchange for each boundary cell $\boldsymbol{x}_b$ can be calculated by a sum over all fluid neighbours

$$\sum \boldsymbol{c}_i \left[f_i(t, \boldsymbol{x}_b) + \boldsymbol{c}_{\bar{i}} f_{\bar{i}}(t, \boldsymbol{x}_b + \boldsymbol{c}_i \Delta x) \right], \tag{6.4}$$

where $\boldsymbol{x}_b + \boldsymbol{c}_i \Delta x$ is the location of a fluid cell. Then, the total force on the boundary can be calculated by summing up all contributions of each boundary cell $\boldsymbol{x}_b$

$$\boldsymbol{F} = \sum_{\boldsymbol{x}_b} \sum \boldsymbol{c}_i \left[f_i(t, \boldsymbol{x}_b) + \boldsymbol{c}_{\bar{i}} f_{\bar{i}}(t, \boldsymbol{x}_b + \boldsymbol{c}_i \Delta x) \right]. \tag{6.5}$$

It must be emphasised that the implementation of Equation (6.5) depends on the applied streaming pattern. The access pattern for the boundary conditions changes according to the streaming pattern, which must be correctly accounted for. Furthermore, Equation (6.5) requires a reduction on the finest level for the particle distribution functions (PDFs). This can potentially involve many simulation cells. Thus, calculating the total force can quickly become a main bottleneck if not implemented carefully. In this work, we use the code generation pipeline and extend the boundary kernels with the functionality to directly calculate the momentum exchange after the boundary PDFs has been updated inside the same compute kernel. The compute kernels can now be generated with and without computing the momentum exchange. The advantage of this idea is that the streaming pattern is automatically accounted for correctly, and the momentum exchange method is calculated on data that has just been loaded from main memory and is, therefore, still in a register. Due to this idea, it is possible to evaluate the drag coefficient in every fine timestep with an overhead of less than 10%. As shown by Geier *et al.* [60], the drag coefficient undergoes high fluctuations, especially in the transition regions from the sub-, to the super-critical region. For this reason, it can be difficult to find a suitable evaluation frequency beforehand that does not alter the results when the final drag coefficient is calculated as the mean value of the instantaneous drag coefficient. The implementation presented here allows us to evaluate the simulation using the highest possible resolution.

The xy-plane of the instantaneous flow field is pictured in Figure 6.5. This result is obtained after 60 000 coarse timesteps and with various Reynolds numbers. As the Reynolds numbers increase, the wake after the round body becomes steeper and steeper, similar to the experimental results pictured in Figure 6.1.

The average drag coefficient is illustrated in Figure 6.3 as a next step. The value for the average drag coefficient is calculated by taking the average of the measured drag coefficient for the last 500 000 fine timesteps. The results are compared to experimental data by Achenbach [175] and Prandtl [179] and two regression functions that Morrison [180] and Almedeij [181] have introduced. Furthermore, our findings are compared to the simulations of Geier *et al.* [60]. Their work was the first to recover the drag crisis of a sphere with the cumulant LBM. While there are many differences between the work of Geier *et al.* due to the different code structures of VirtualFluids [148] and waLBerla, the major difference is the interpolation order between grids of different resolutions. In their work, a compact interpolation method applies a fourth-order interpolation for the grid transition [149]. In contrast, a simpler explosion method is implemented in waLBerla. Nevertheless, Figure 6.3 reveals that our simulation setup can also predict the drag crisis.

The average drag coefficient in Figure 6.3 is calculated on the instantaneous drag coefficient, shown in Figure 6.4. It reveals that the drag coefficient shows high fluctuations. This phenomenon has also been reported in the work of Geier *et al.*. However, it seems that the fluctuations presented here are higher. Nevertheless, the drag coefficient shows fluctuations in a lower flow regime for higher Reynolds numbers.

6.4 Conclusion

This summarises the developments of LBMpy and waLBerla in the efforts of this thesis. It can be shown that complex phenomena like the drag crisis can be predicted numerically using statically refined grids on GPGPUs. Here, we have employed a relatively simple grid transition technique. However, the results are in good agreement with the literature. This

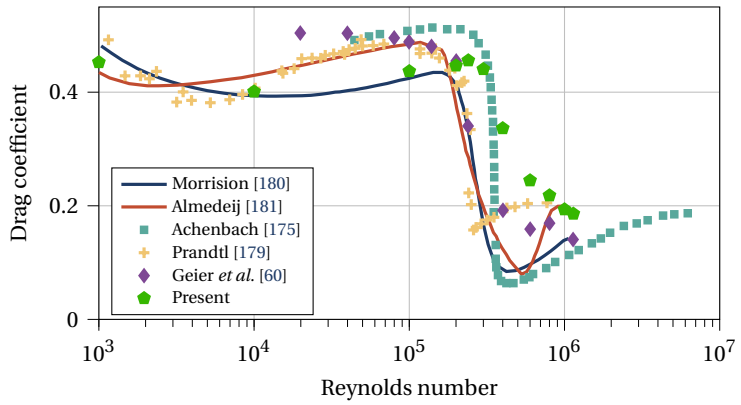


Figure 6.3: Average drag coefficient plotted against the Reynolds number Re. The average drag coefficient is calculated by averaging the drag coefficient of the last $5 \cdot 10^5$ fine timesteps.

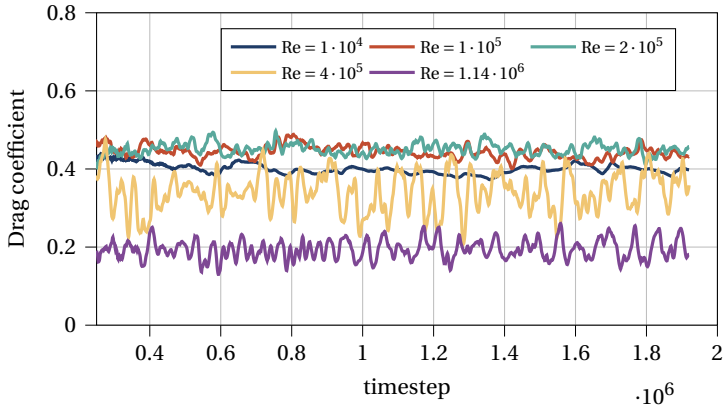
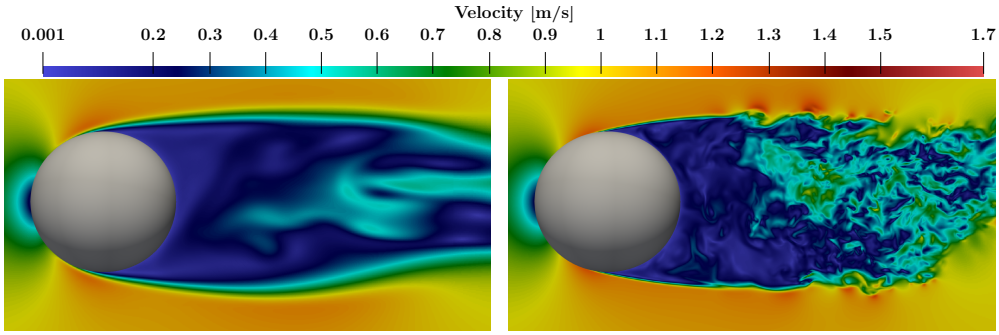


Figure 6.4: Temporal evolution of the measured drag coefficient c_D for Reynolds numbers $\mathrm{Re} \in \{1 \cdot 10^4, 1 \cdot 10^5, 2 \cdot 10^5, 4 \cdot 10^5, 1.14 \cdot 10^6\}$. The instantaneous drag coefficient shows strong fluctuations, which causes an uncertainty in the average drag coefficient shown in Figure 6.3. A similar phenomenon has been reported in the findings of Geier *et al.* [60]. However, it must be noted that the fluctuations in Geier's work seem smaller than ours.

goes along with recent findings of Astoul *et al.* [169, 182]. Their studies showed that the non-hydrodynamic modes cross the grid interface and introduce spurious contributions to the vorticity. It was concluded that these non-hydrodynamics occur independently of the details of the grid transfer algorithm. Hence, the suggestion is to use an appropriate collision method to account for the non-hydrodynamic contributions. Recent studies assume that the cumulant collision operator might be a good candidate in this matter [183].



(a) Re = 1000

(b) Re = 10 000

(c) Re = 100 000

(d) Re = 200 000

(e) Re = 300 000

(f) Re = 400 000

(g) Re = 600 000

(h) Re = 800 000

(i) Re = 1 000 000

(j) Re = 1 140 000

Figure 6.5: Instantaneous velocity field of the simulation of a sphere under various Reynolds numbers. The simulation is shown at a cut through the xy-plane after 60 000 timesteps on the coarsest mesh. Increasing the Reynolds number shows a decreasing flow separation angle after the spherical body.

CHAPTER 7

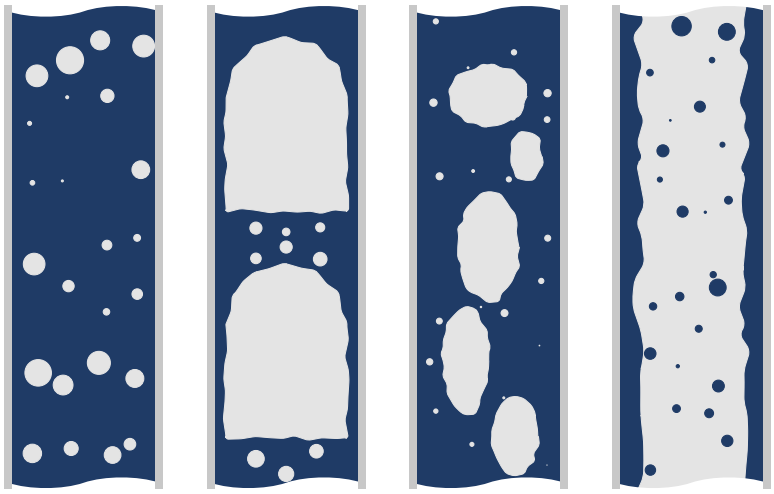
DISPERSED FLOW DYNAMICS

Contents

In this chapter, the numerical results for dispersed flows are presented. After a brief introduction in Section 7.1, an analysis of Taylor bubbles, which occur in the slug flow regime, is performed in Section 7.2. At first, the work in this thesis aims to reproduce existing literature to validate the implementation of the conservative Allen-Cahn model in WALBERLA. *This base case presents the dynamics of a Taylor bubble in an annular pipe configuration. Afterwards, the pipe geometry is changed in subsequent simulations. Here, two specific changes are made. First, the inner tube of the pipe is moved in its position in Section 7.2.2, and second, it is changed in size in Section 7.2.3. Finally, we present a conclusion in Section 7.3. The results presented here follow the author's contribution to the published article [2], which was published as part of the thesis.*

7.1 Brief Introduction

Numerical prediction of fluid flow in confined systems plays a crucial role in the natural gas industry. The reason is that pipelines are the primary transport method for fluids and must be designed for efficiency and safety. A challenging aspect arises from the fact that fluid flow in these systems manifests in numerous characteristic patterns known as flow regimes. These regimes depend on the geometry of the pipe system, the physical properties and the flow rates of the fluids [184]. The four most important flow regimes are illustrated in Figure 7.1. The bubble flow regime shown in Figure 7.1a is observed for relatively low gas velocities and gas-liquid ratios. As these increase, bubble coalescence is enhanced, and a slug flow regime is generated in which liquid slugs separate a train of elongated gas bubbles. The slug flow regime is pictured in Figure 7.1b. The instability of the elongated or Taylor bubbles leads to the churn flow regime (compare Figure 7.1c), and a further increase in gas velocity results in the annular flow regime (compare Figure 7.1d)[2, 184]



(a) Bubble flow (b) Slug Flow (c) Churn Flow (d) Annular Flow

Figure 7.1: Illustration of the four main flow regimes in a pipe flow with liquid in gas phases. Illustration recreated from [184]

The focus of this section lies on the investigation of pipe flows in the slug flow regime. The elongated bubbles which occur in this regime are often referred to as Taylor bubbles. They are bullet-shaped bubbles that occupy almost the entire cross-section of a pipe [185]. There are numerous applications in which Taylor bubbles occur. Prominent examples are geothermal energy production, chemical plants, emergency events within nuclear reactors or cardiovascular systems [185–187]. Due to its prominence in natural, medical and industrial applications, the description of Taylor bubble dynamics has received significant attention in the literature from theoretical [188], experimental [189–191] and numerical points of view [187, 192, 193]. Various numerical approaches have been used to analyse the flow dynamics of Taylor bubbles. Among these, one of the most popular methods is the volume-of-fluid method (VoF), which has been applied in various studies [194–196]. Within the commercial code TransAT© [1] Taylor bubbles have been simulated using the level-set method [192]. Furthermore, the conservative Allen-Cahn model (CACM) has been employed by Mitchell *et al.* [117] to analyse Taylor bubbles both in tubular and annular pipe configurations.

In the scope of this thesis also, a comparative study between a free surface approach and the CACM has been conducted [3]. Within this study, both methods have been used to simulate Taylor bubbles in a cylindrical setup. It was shown that both methods can capture the flow dynamics in the slug flow regime correctly in a simplified setup. Herein, a single-forming Taylor bubble's shape and the rise velocity were analysed and compared with the literature. While the free surface method could deliver stable results for lower resolutions, similar resolutions were needed for satisfactory accuracy for both methods. In the free surface approach, only the liquid phase is simulated. In contrast, the gas phase is neglected, and missing information at the interface is reconstructed with free surface boundary conditions [88]. This reduces the memory and computational requirements due to the smaller region which is simulated. On top of that, only a single set of populations is used since no second partial differential equation (PDE) needs to be solved. However, the free surface boundary conditions increase the algorithmic complexity. In particular, phenomena like bubble splitting or merging require a complex bubble model that considers these scenarios. Such a model is hard to apply in highly parallel environments [197]. Since Taylor bubble simulations in annular pipe configurations require highly resolved grids and a particular focus lies on the dynamics of the gas phase, it was plausible to choose the CACM for the simulations in this thesis.

7.2 Taylor Bubble Dynamics

The implementation of the CACM follows the equations presented in Chapter 3. As a first step, the CACM implemented in waLBerla using the code generation facility is validated. This is done based on the work presented by Mitchell *et al.* [118]. As part of the numerical validation, additional insights are gathered by a small parameter study on the mobility parameter M. Since the mobility is directly linked to the relaxation rate of the Allen-Cahn solver (compare Equation (3.20)), it is a value of high interest because it poses implications on the stability of the solver. Building on the validation case, extended setups are analysed in Sections 7.2.2 and 7.2.3. Herein, the effect of pipe eccentricity and diameter ratio was

[1] https://transat-cfd.com/

analysed, and the impact on the rise velocity and shape of Taylor bubbles was reported. The mesh refinement capabilities of WALBERLA are not yet compatible with the CACM. Hence, we produced the results of this chapter and Chapter 8 using a uniform domain. The extension of the refinement algorithm with the CACM will be future work.

7.2.1 Experimental Validation

Three annular pipe setups have been introduced in the work of Das *et al.* [188]. In this section, setup *C* (compare table 2 from reference [188]) is analysed. This problem was also studied by Mitchell *et al.* [118]. In this setup, the diameter of the inner tube is $d_{\text{inner}} = 0.0127\,\text{m}$ and the diameter of the pipe is $d_{\text{outer}} = 0.0254\,\text{m}$. The fluid properties with their respective dimensionless numbers are shown in Table 7.1. Notably, the Cahn number Cn, which defines the interface width of the phase field model, is fixed here. It was thus added to the table. The reference time

$$t^* = \sqrt{\frac{d_\text{h}}{g}} \tag{7.1}$$

and the velocity scale

$$U_0 = \sqrt{g(d_{\text{inner}} + d_{\text{outer}})} \tag{7.2}$$

are introduced to compare the physical experiments with the numerical results gathered in this section. From the reference time and the timestep t, the normalised time $t_\text{b}^* = {}^{t}/_{t^*}$ is calculated. Herein, d_h is the hydraulic diameter for an annular pipe, and g is the gravitational acceleration. With the velocity scale, the results of this study will be presented in the form of the Froude number,

$$\text{Fr} = \frac{u_\text{rise}}{U_0}, \tag{7.3}$$

which defines the ratio of the rise velocity u_rise to the velocity scale U_0 which gives a measure of the external force field. The other important dimensionless parameters here are the Eötvös number Eo, which relates gravitational to capillary forces, and the Morton number Mo, which is the ratio of viscous to surface tension forces.

Table 7.1: Fluid properties including density, ρ, viscosity, μ, and surface tension, σ, as well as the dimensionless parameters defined as the Eötvös, Eo, Morton, Mo, Cahn Cn, and inverse viscosity, N_f, number. Note that d_h is the hydraulic diameter for an annular pipe.

Parameter	Liquid	Gas	Units
ρ	998	1.2047	$\text{kg}\,\text{s}^{-3}$
μ	$1.002 \cdot 10^{-3}$	$1.8205 \cdot 10^{-5}$	$\text{kg}\,\text{m}^{-1}\,\text{s}^{-1}$
σ	0.072 86		$\text{kg}\,\text{s}^{-2}$
	Formulation	Value	
Eo	$(\rho_\text{H} - \rho_\text{L}) g d_\text{h}^2 / \sigma$	21.6468	[-]
Mo	$\mu_\text{H}^4 (\rho_\text{H} - \rho_\text{L}) g / (\rho_\text{H}^2 \sigma^3)$	$20.5587 \cdot 10^{-11}$	[-]
Cn	W/D	0.039	[-]
N_f	$(\text{Eo}^3/\text{Mo})^{0.25}$	4462	[-]

The total domain is formed by a rectangular prism of size $(n_x \times n_y \times n_z) = (D \times 15D \times D)$, where D is the number of lattice units to resolve the diameter of the pipe d_{outer}. The pipe containing an inner tube to form the annulus is placed inside the prism. For this study, a specific initialisation of the bubble was used, which takes the form

$$x_{\text{int}}(y,z) = d_f \sqrt{r_1^2 - \left(r_2 - \sqrt{(y-y_c)^2 + (z-z_c)^2}\right)}, \tag{7.4}$$

where d_f is the dilation factor, r_1 is the radius of the inner tube, and r_2 is the distance from the annular pipe axis to the centre of the inner tube. The subscript, c, was used to designate the central location of the annular pipe. The top section of this is modified with a sinusoidal function. This setup is shown in Figure 7.2a. It is replicated from the studies of Mitchell *et al.* [198]. Further, it must be noted that the initialisation plays an important role in the stability of the setup. A simple torus shape for the initialisation would result in strong dynamics at the beginning, eventually crashing the simulation. Different strategies exist to prescribe a smooth transition between the phases after initialising the bubble with a sharp interface. For example, Mitchell used a fixed number of timesteps in which he only executed the Allen-Cahn solver in his thesis [79]. In this work, a Gaussian filter is employed, similar to the work of Li *et al.* [199].

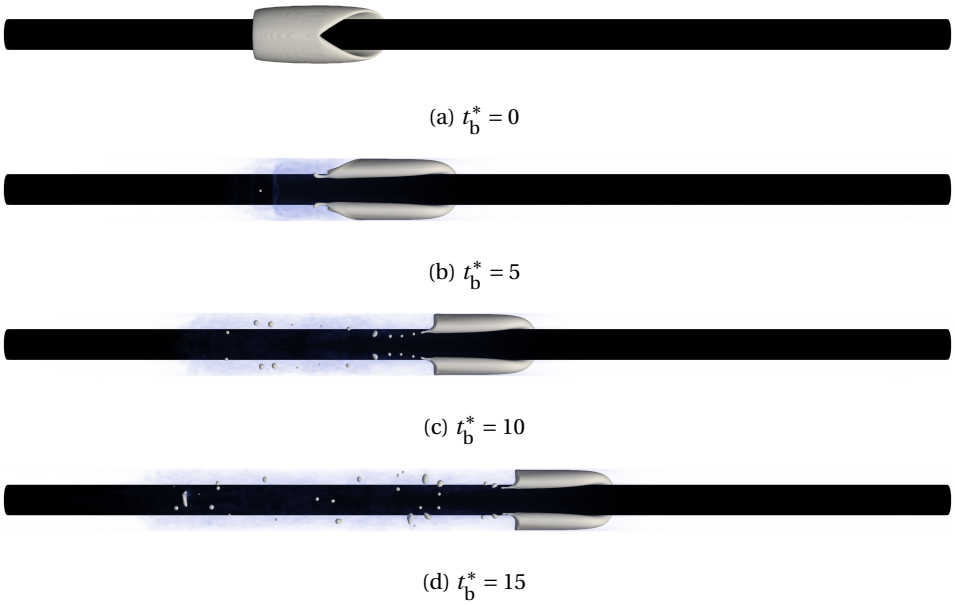
A lattice resolution of $D = 128$ lattice cells and a reference time of $t^* = 16 \cdot 10^3$ was specified for the validation case. The numerical viscosity was kept constant when analysing higher resolution scenarios, resulting in increased dimensionless time. The resolutions examined here were $D = \{128, 160, 192, 224, 256, 288\}$ with $t^* = \{16, 25, 36, 49, 64, 81\} \cdot 10^3$, respectively. The reference time and dimensionless parameters are sufficient to define the physical system properties for each simulation. However, the mobility parameter remains undefined and can be chosen freely.

The temporal evolution of the bubble as it propagates through a stagnant fluid is shown in Figure 7.2. Here, the development of the liquid bridge and the turbulent wake region can be observed. In the dynamics of the liquid bridge, smaller satellite bubbles form, as typical for slug flows. It emphasises the rich dynamics which occur in these flow scenarios. Besides that, the figure provides a view of the region of influence that the Taylor bubble has in the tube, with a long wake region developing and the initially stagnant fluid near the bubble nose draining towards the liquid film and bridge. During each simulation, the Taylor bubble's centre of mass, $\boldsymbol{x}_*$, was tracked. In addition to this, the integral velocity was also determined during the simulation through,

$$\boldsymbol{U}_* = \frac{\sum_\Gamma (1 - \phi(\boldsymbol{x}))\boldsymbol{u}}{\sum_\Gamma (1 - \phi(\boldsymbol{x}))}, \tag{7.5}$$

where Γ represents the set of lattice nodes within the Taylor bubble.

This data was tracked with a high frequency during simulations, and as such, smoothing is necessary to remove noise and fluctuations. Thus, a Savitzky-Golay filter was applied, which locally mapped polynomials to smooth the positional data [200]. Calculating the first- and second-order derivatives from the bubble position results in the velocity and acceleration as shown in Figure 7.3. Here, an example is given for the centre of mass, which was captured along the axis of the annular pipe. The Froude number Fr is also given in Figure 7.3b. As pictured, the initialisation results in high Froude number oscillations. Afterwards, the bubble progresses, and a smooth, steady state is reached. The positional data indicates a simulation run time of $15t^*$, whereas only t^* to $14t^*$ are indicated on the velocity plot. This is a result of averaging and smoothing conducted on the position data.



(a) $t_b^* = 0$

(b) $t_b^* = 5$

(c) $t_b^* = 10$

(d) $t_b^* = 15$

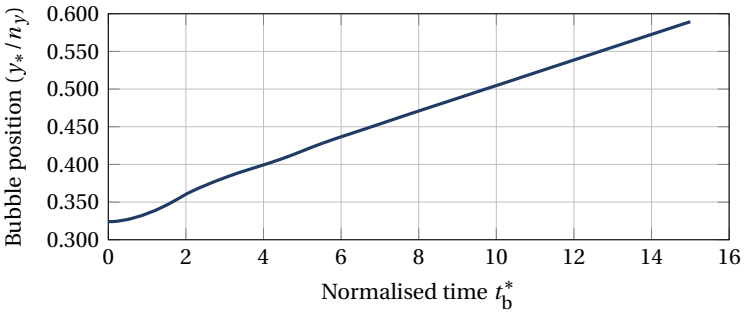
Figure 7.2: Evolution of a Taylor bubble through a concentric pipe with a mobility of $M = 0.05$. The initialisation of the bubble is shown in (a). In (b) at $t_b^* = 5$, the bubble forms a liquid bridge which lets liquid directly pass when the bubble is rising. The vorticity of the liquid is shown in blue colour. After forming the liquid bridge, the rising speed of the bubble increases and in (c) at $t_b^* = 10$, the bubble has already reached the final velocity (compare Figure 7.3). In (d), the last simulation step is shown at $t_b^* = 15$.

The steady-state velocity for various lattice resolutions is presented in Figure 7.4. The mobility parameter range was tested with $M \in \{0.02, \ldots, 0.1\}$. Notably, the mobility parameter significantly influences the simulations' stability. The reason is the inverse relation with the relaxation rate ω_ϕ of the Allen-Cahn solver as given by Equation (3.20). In our simulations, mobility parameters below 0.05 lead to instabilities. However, the steady-state solution of the velocity shows no significant influence by the mobility. In Figure 7.4, the results were all in the same range by a variation with 1 %. Hence, it can be concluded that the mobility parameter influences the stability of the simulations much greater than the accuracy in the cases analysed here. As a result, the following sections are investigated using a mobility of $M = 0.05$.

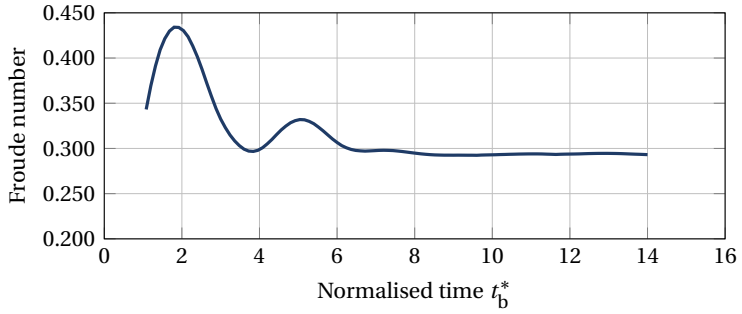
To find closure relationships for the dynamics of Taylor bubbles in annular configurations, analytic equations were developed in the work of Mitchell and Leonardi [118] to estimate the rise velocity based on the dimensionless numbers. In particular a relationship based on Eo and N_f was identified by,

$$\mathrm{Fr}(\mathrm{Eo}, N_f) = \frac{\frac{0.383}{[1+(1.4078/\mathrm{Eo})^{0.6442}]^{1.5381}}}{\left(1 + \left[\frac{N_f}{3.6321(1+(-1.5753/\mathrm{Eo}))^{2.7688}}\right]^{-1.1635}\right)} = 0.299 \tag{7.6}$$

The results shown in Figure 7.4 underpredict the rise velocity calculated by Equation (7.6) if the mean velocity of all simulations is taken. However, it still shows a close alignment of the numerical results in comparison to the closure relationship developed by Mitchell and Leonardi [118]. This highlights the effectiveness of the simulations conducted in the scope of this thesis. Nevertheless, it must also be noted that experimental results



(a) Taylor bubble centre of mass tracked during simulation.



(b) Froude number derived from centre of mass.

Figure 7.3: A Savitzky-Golay filter applied to the bubble position in (a) to obtain the bubble velocity in (b) for a simulation constructed with an outer diameter of 192 cells and a mobility value of $M = 0.05$.

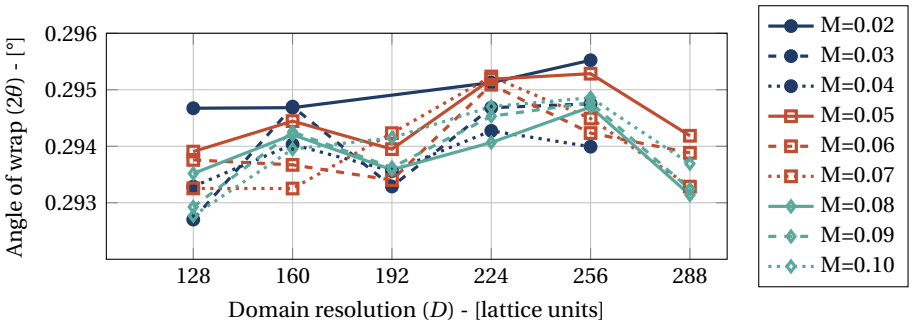


Figure 7.4: Convergence analysis of the dimensionless rise velocity of an annular Taylor bubble. A range from 0.02 to 0.1 was investigated for the mobility parameter. Note the scale of the y-axis, indicating limited impact on macroscopic bubble velocity but reduced stability for lower values of M.

obtained by Das *et al.* [188] predict a Froude number of 0.28. In comparison to this, the simulations had an error of approximately 6 % similar to the estimation predicted by Equation (7.6). Therefore, the analysis still validates the modelling approach because it was possible to closely capture the evolution of the Taylor bubble compared to Mitchell and Leonardi [118]. Additionally, the effect of the mobility parameter was analysed, which made valuable contributions to accuracy and stability under varying parameters.

Another important aspect when analysing Taylor bubbles in an annulus setup is the wrap angle θ of the bubble about the inner tube. Thus, Figure 7.5 provides the steady-state wrap angle of the Taylor bubble about the inner pipe. It was determined by creating a contour of ϕ_0 in the plane perpendicular to the pipe axis at the bubble's centre of mass. Then, the wrap angle can be obtained from the discrete data using a simple iterative method. To better understand how the wrap angle is defined in this work, an illustration is provided with Figure 7.6. In the illustration, an example contour is shown, with tangent lines drawn to indicate the wrap angle of the Taylor bubble.

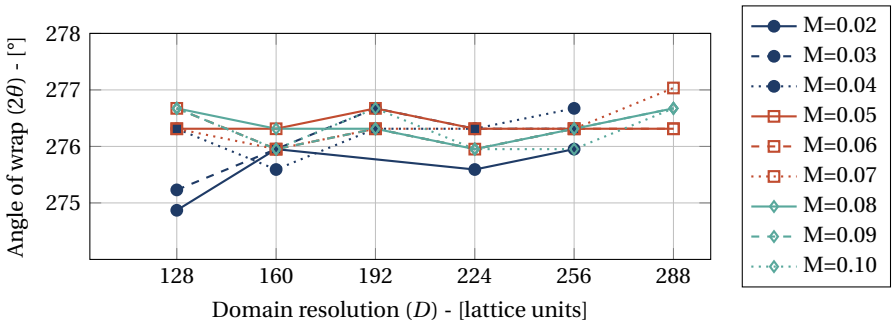


Figure 7.5: Convergence analysis of the shape of a Taylor bubble. The shape of the Taylor bubble is analysed through the angle of wrap 2θ. A range from 0.02 to 0.1 was investigated for the mobility parameter.

The results in this section show that the tested resolutions consistently validate the simulations. Furthermore, the diameter of the pipe is resolved using 192 lattice cells, which leads to a spatial resolution of $\Delta x = 1.323 \cdot 10^{-4}$. This gives a reasonable balance between computational requirements for the increased complexity which arises from the variability of the inner tube configuration.

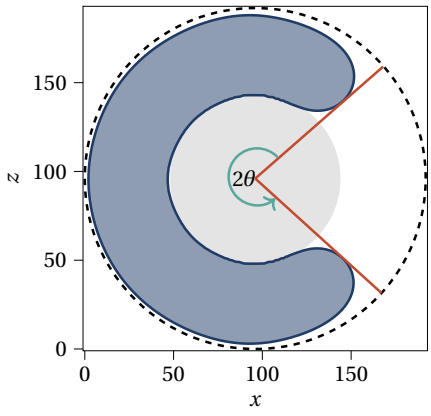


Figure 7.6: Example interface contour of a Taylor bubble at its centre of mass. The angle of the wrap about the pipe is measured with the contour. It is defined as indicated by the green circle arc.

7.2.2 Variable Inner Pipe Eccentricity

In this section, the impact of the inner tube location on the flow dynamics of the Taylor bubble is analysed. The shape of the inner tube is not changed, and the eccentricity parameter states the variation in its location. A schematic illustration is given with Figure 7.7 to provide a better understanding of how the eccentricity is measured. With the notation used in this figure, the degree of eccentricity can be defined as

$$\varepsilon = \frac{c_i - c_o}{r_{\text{outer}} - r_{\text{inner}}}. \tag{7.7}$$

Using this, the eccentricity values investigated in this section consist of $\varepsilon \in \{0.1, 0.2, \ldots, 0.9\}$. The bubble rise velocity and the angle of wrap about the inner pipe are recorded for each scenario.

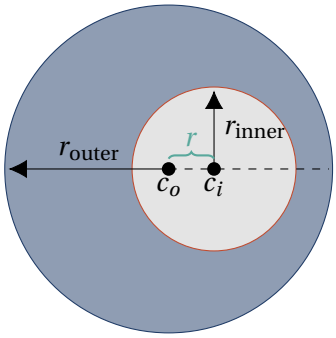


Figure 7.7: Schematic cross-section of an eccentric annular pipe in which the central tube is offset by a distance r.

The simulation's initialisation is done the same way as in the previous section. Because of varying eccentricity parameters, finding suitable initial bubble shapes is hard. From our experience, a naive initialisation of the Taylor bubble usually leads to initial instabilities. Therefore, the simulations are not conducted using the changed inner tube position at the initial position of the Taylor bubble. Instead, the inner tube has an eccentricity $\varepsilon = 0$ at the initial position. Then, the Taylor bubble enters a transition region of length $4D$ where the inner tube location changes to its target eccentricity. The pipe bend was incorporated with a half-period of a cosine function, which allows for a smooth transition. A visual representation of the simulation setup is given in Figure 7.8. In this figure, an eccentricity $\varepsilon = 0.5$ is used. The transition region is highlighted to better understand how it is applied. For all simulations, the run-time was increased to $20t^*$. This is necessary to observe a stable Taylor bubble after the transition region.

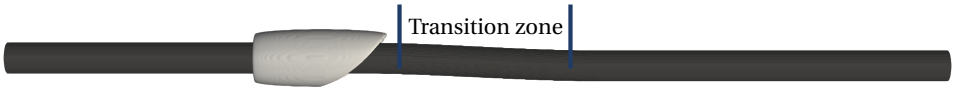
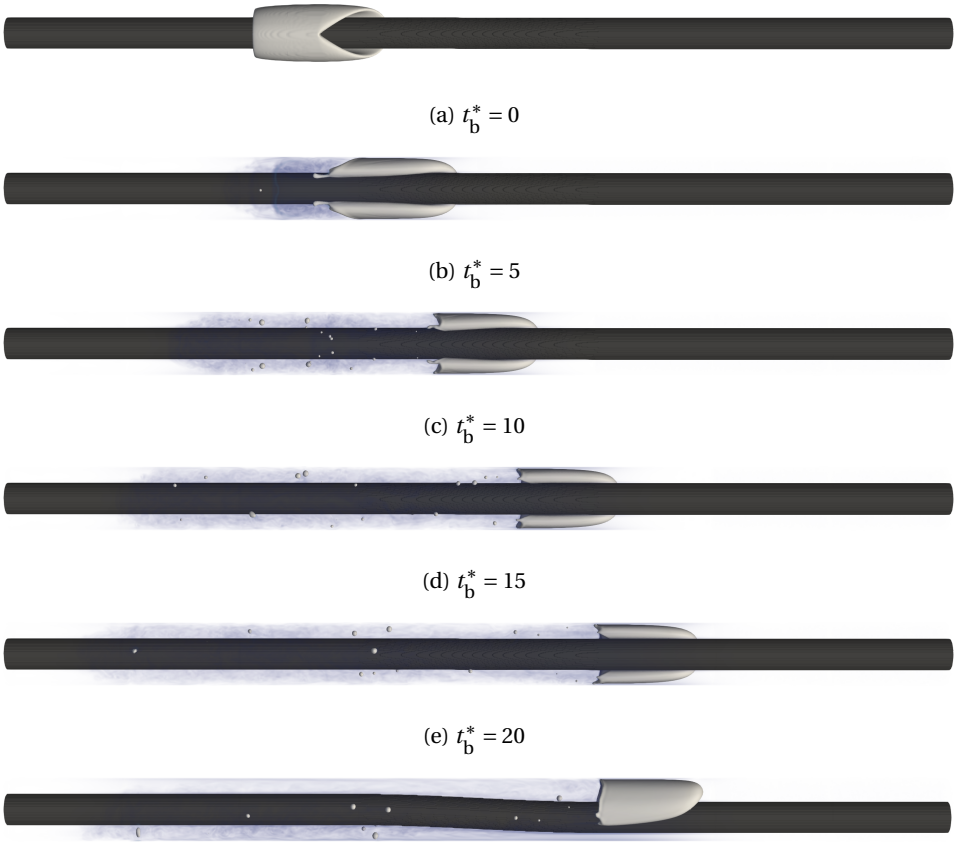


Figure 7.8: Geometry of the inner tube in the simulation domain for eccentricity value of $\varepsilon = 0.5$ with the transition region indicated. Note that gravity is applied along the pipe axis so that the case is representative of a vertical pipe configuration.

The Taylor bubble dynamics moving from a concentric to an eccentric pipe configuration is illustrated in Figure 7.9. It can be observed that the bubble smoothly progresses through the transition zone until it reaches its final terminal velocity and shape. Due to the relatively high velocity at the boundary of the liquid bridge, the formation of small satellite bubbles can be observed. Building on the results of the previous section, both the rise velocity and the angle of wrap are measured similarly. Like this, it is possible to assess the impact of eccentricity quantitatively. Understanding the impact eccentricity has on Taylor bubble dynamics can be used to reduce the uncertainty of pipe network designs for numerous applications.



(a) $t_b^* = 0$

(b) $t_b^* = 5$

(c) $t_b^* = 10$

(d) $t_b^* = 15$

(e) $t_b^* = 20$

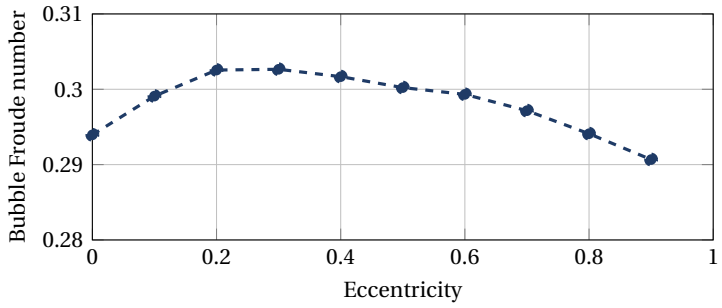
(f) $t_b^* = 20$ (rotated by 90°)

Figure 7.9: Evolution of a Taylor bubble through a pipe that transitions from concentric to an eccentricity value of 0.5. The initialisation of the bubble is shown in (a). In (b) at $t_b^* = 5$, the bubble forms a liquid bridge which lets liquid directly pass when the bubble is rising. The vorticity of the liquid is shown in blue colour. In (c) at $t_b^* = 10$, the Taylor bubble propagates through the transition region of the pipe. Hence, the shape of the bubble is changing through this section. In (d) at $t_b^* = 15$, the Taylor bubble has just passed the transition region and reaches its final shape. In (e) and (f), the final shape of the bubble is shown at $t_b^* = 20$.

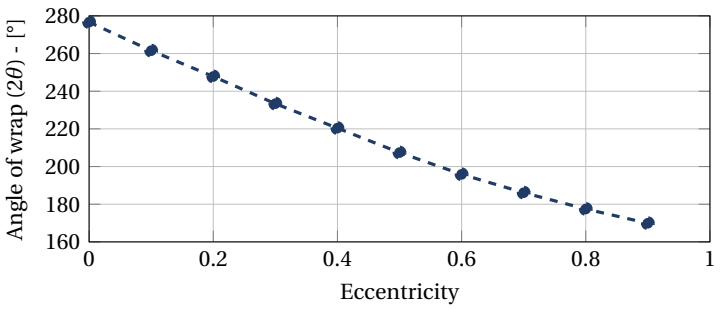
The steady-state rise velocity obtained for various eccentricity values is given in Figure 7.10a. Clearly, the rise velocity of the Taylor bubble is not strongly influenced by the changed geometry. The most significant divergence in comparison to the rise velocity of the base geometry can be observed for eccentricity values of 0.2 and 0.3. At these points, the bubble Froude number is increased by approximately 3 %. As the eccentricity values

increase, the bubble rise velocity reaches values similar to those in the base case. The differences here are approximately 1 % for $\varepsilon = 0.9$. This magnitude of variation in rise velocity may be acceptable for certain circumstances. However, understanding it allows for a reduction in uncertainty when looking to predict pipe system characteristics such as pressure drop.

Additionally, the angle of wrap was analysed and is illustrated in Figure 7.10b. A smooth decrease can be observed as the inner tube is shifted more and more towards the outer wall. As the inner tube moved into close proximity to the outer pipe wall, the bubble rise velocity reduced to a value below that of the concentric configuration (see Figure 7.10a). It is expected that this correlates with a widening of the Taylor bubble nose in the larger cavity available at higher levels of eccentricity. This would lead to an increase in viscous drag as the shape of the bubble nose progresses away from the elliptic shape commonly referenced for annular Taylor bubbles to the spherical shape assumed in the analysis of the tubular analogue [188]. The work conducted here is the first instance in which the dependence of Taylor bubble rise velocity on various levels of eccentricity has been shown in vertical annular systems.



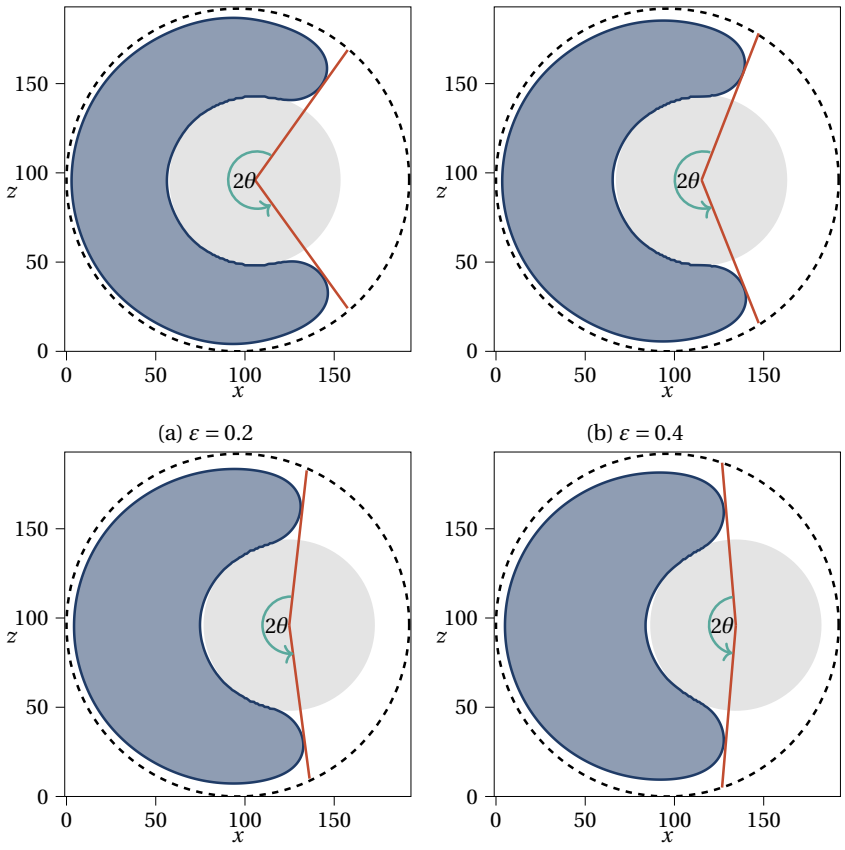
(a) Rise Velocity



(b) Angle of Wrap

Figure 7.10: The effect of pipe eccentricity on the propagation of an annular Taylor bubble in a water-filled, annular conduit. The rise velocity is given in (a) and the angle of wrap in (b).

Finally, the Taylor bubble shape progression for various eccentricity values is shown in Figure 7.11. The angle of wrap is included in each subfigure to indicate how it is determined. Here, the increased width of the bubble section opposite the liquid bridge for high eccentricity is clear, and the evident increase in viscous drag was observed in a decreasing rise in velocity. From these results, it can be concluded that a minor shift in the inner pipe away from the central axis of the outer pipe causes a rise in bubble velocity. However, after reaching a maximum, the viscous drag caused by the increased width of the bubble dominates and slows its velocity.



(a) $\varepsilon = 0.2$ (b) $\varepsilon = 0.4$

(c) $\varepsilon = 0.6$ (d) $\varepsilon = 0.8$

Figure 7.11: Steady-state wrap angle, 2θ, for an annular Taylor bubble at eccentricity values of (a) 0.2, (b) 0.4, (c) 0.6, (d) 0.8.

7.2.3 Variable Pipe Diameter Ratio

The impact of the diameter of the inner tube on the dynamics of the Taylor bubble is analysed in this section. The steady-state bubble rise velocity and the final shape (by evaluating the angle of wrap) are investigated. Early correlations of bubble rise velocity indicated dependence on the inner pipe size, with non-dimensional velocities often formulated as

$$\mathrm{Fr} = \frac{U}{\sqrt{g(d_{\mathrm{inner}} + d_{\mathrm{outer}})}}. \tag{7.8}$$

This has previously been used to state a Froude number applicable to water-air flow for various pipe diameter ratios. Here, we further test its applicability. A graphical illustration of how the pipe ratio is quantified in this work is given in Figure 7.12. Following the notation of the figure, the pipe ratio can be defined by

$$d^* = \frac{r_{\mathrm{inner}}}{r_{\mathrm{outer}}}. \tag{7.9}$$

This means the base case can be recovered by using $d^* = 0.5$. In the analysis here, various pipe ratios have been tested in the range of $d^* \in \{0.0, 0.05, \ldots, 0.6\}$. Building on the investigations conducted in the previous section, the same fluid parameters were applied. Therefore, a mobility of $M = 0.05$ is also used. A schematic cross-section of the pipe with

a reduced inner tube diameter is given in Figure 7.12. In this illustration, a diameter of $d_{\text{inner}} = 0.00635\,\text{m}$ is applied for the inner tube. Using a resolution of $\Delta x = 1.323 \cdot 10^{-4}$ leads to a diameter of 48 lattice cells of the inner tube in the simulation. The illustrated pipe ratio is $d^* = 0.25$.

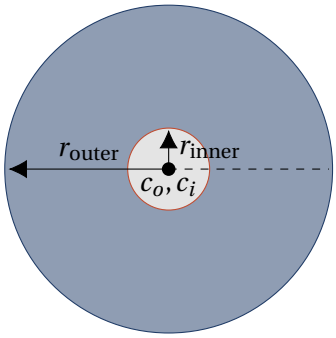


Figure 7.12: Schematic cross-section of an annular pipe in which the radius of the central tube, r_{inner}, was the dimension varied to obtain the pipe diameter ratios analysed. In this schematic cross-section, a pipe ratio of $d^* = 0.25$ is visualised.

Like the eccentricity simulations shown in Section 7.2.2, the Taylor bubble is initialised at a region with a pipe ratio of $d^* = 0.5$. Then the Taylor bubble finds a transition region with the length of 4D. In this region, the pipe configuration changes to its target pipe ratio. An example of the setup is shown in Figure 7.13 for a pipe ratio of $d^* = 0.25$. The transition zone is marked in the figure for better understanding.

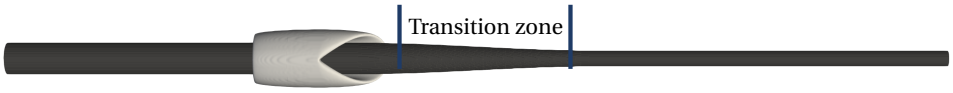
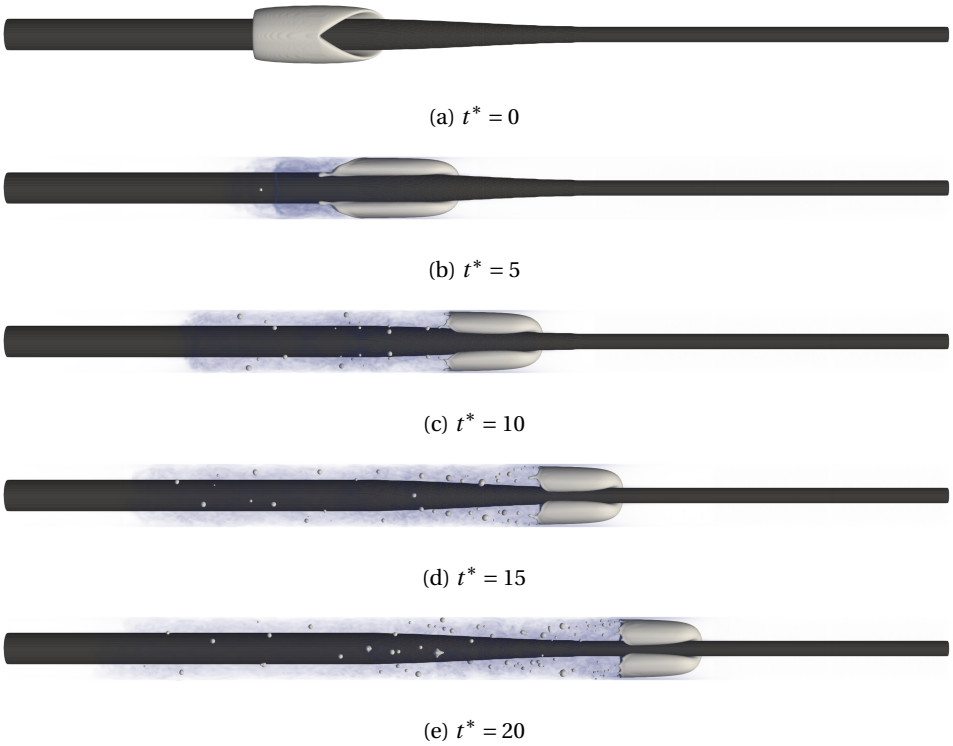


Figure 7.13: Geometry of the inner tube in the simulation domain for a pipe ratio of $d^* = 0.25$ with the transition region indicated. Note that gravity is applied along the pipe axis so that the case is representative of a vertical pipe configuration.

The temporal evolution of the Taylor bubble in an annular pipe configuration with a pipe ratio $d^* = 0.25$ is shown in Figure 7.14. It highlights the adaption of the angle of wrap that the Taylor bubble undergoes. During the transition to the reduced pipe ratio, the wrap angle increases smoothly, leading to a Taylor bubble that almost wraps around the whole inner tube. Thus, a very small liquid bridge remains. Furthermore, the simulation shows an increased generation of satellite bubbles into the wake region. Additionally, the length of the Taylor bubble shrinks.

In Figure 7.15a, the variation of the Froude number with changing pip ratio is given. Furthermore, Equation (7.6) is used to predict the Froude number analytically. It is possible to apply Equation (7.6) by incorporating the pipe ratio d^* through the Eötvös number. In this manner, a close agreement can be found between the simulation results and the predicted Froude numbers. The work of Mitchell and Leonardi [118] only validated the closure relation with a pipe ratio of $d^* = 0.5$. Thus, the insights gathered in this thesis's scope validate their approach's robustness and indicate a good agreement in the range of $d^* \in \{0.0, 0.05, \ldots, 0.6\}$. In Figure 7.15a, a smooth evolution of the steady-state rise velocity can be observed between $d^* \in \{0.15, 0.2, \ldots, 0.6\}$. Values lower than $d^* = 0.15$ show a particular jump in predicting the Froude number. This can be explained by the fact that



(a) $t^* = 0$

(b) $t^* = 5$

(c) $t^* = 10$

(d) $t^* = 15$

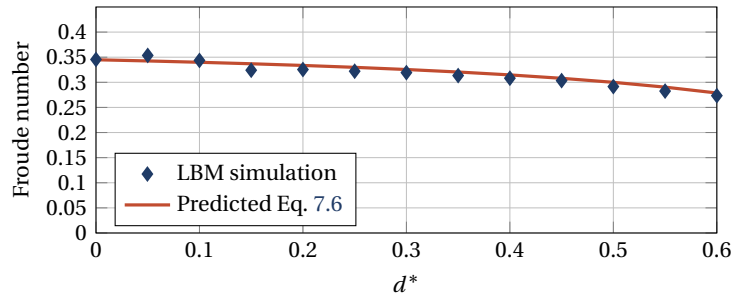
(e) $t^* = 20$

Figure 7.14: Evolution of a Taylor bubble through a pipe that transitions from a pipe ratio of $d^* = 0.5$ to a pipe ratio of $d^* = 0.25$. The initialisation of the bubble is shown in (a). In (b) at $t_b^* = 5$, the bubble forms a liquid bridge which lets liquid directly pass when the bubble is rising. The vorticity of the liquid is shown in blue colour. In (c) at $t_b^* = 10$, the Taylor bubble propagates through the transition region of the pipe. Hence, the shape of the bubble is changing through this section. In (d) at $t_b^* = 15$, the Taylor bubble has just passed the transition region and reaches its final shape. In (e), the final shape of the bubble is shown at $t_b^* = 20$.

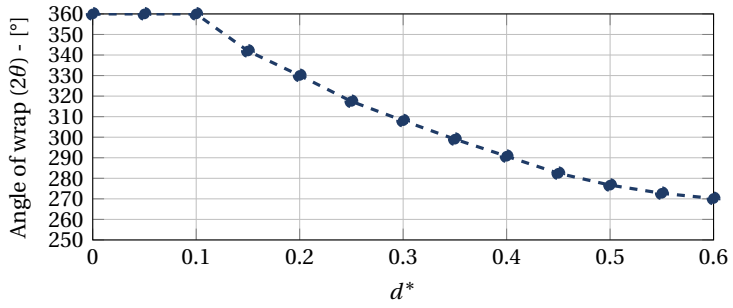
the liquid bridge can no longer be observed in these cases. Instead, the Taylor bubble encloses the inner tube entirely. The missing ability to observe a liquid bridge might come from shortcomings of the diffusive interface model. However, further investigations are necessary to give clear insight into the evolution of Taylor bubbles in annular pipe configuration with decreasing inner tube diameter. Nevertheless, close agreement to Equation (7.6) can be observed also for these cases. Finally, it is important to note that cases of $d^* > 0.6$ were excluded due to numerical instabilities. Also, future work needs to be done for these cases to gain reliable numerical simulations.

In Figure 7.15b, the variation of the wrap angle with changing pip ratio is given. It illustrates again that no liquid bridge is present in cases of $d^* \leq 0.1$. In the figure, a wrap angle of 360° indicates a total wrap of the Taylor bubble around the inner tube. This means that the two sides of the bubble merge. The velocity trend's observed variation was shaped by this factor, highlighting a constraint in the model's applicability within practical computational limits.

Finally, a visual representation of the variation in the cross-sectional shape of the Taylor bubbles for varying annular pipe diameter ratios is given in Figure 7.16. The liquid bridge between the Taylor bubble and the outer wall can be observed clearly. For lower diameter ratios, the liquid bridge vanishes. In contrast, for higher diameter ratios, the



(a) Rise Velocity



(b) Angle of Wrap

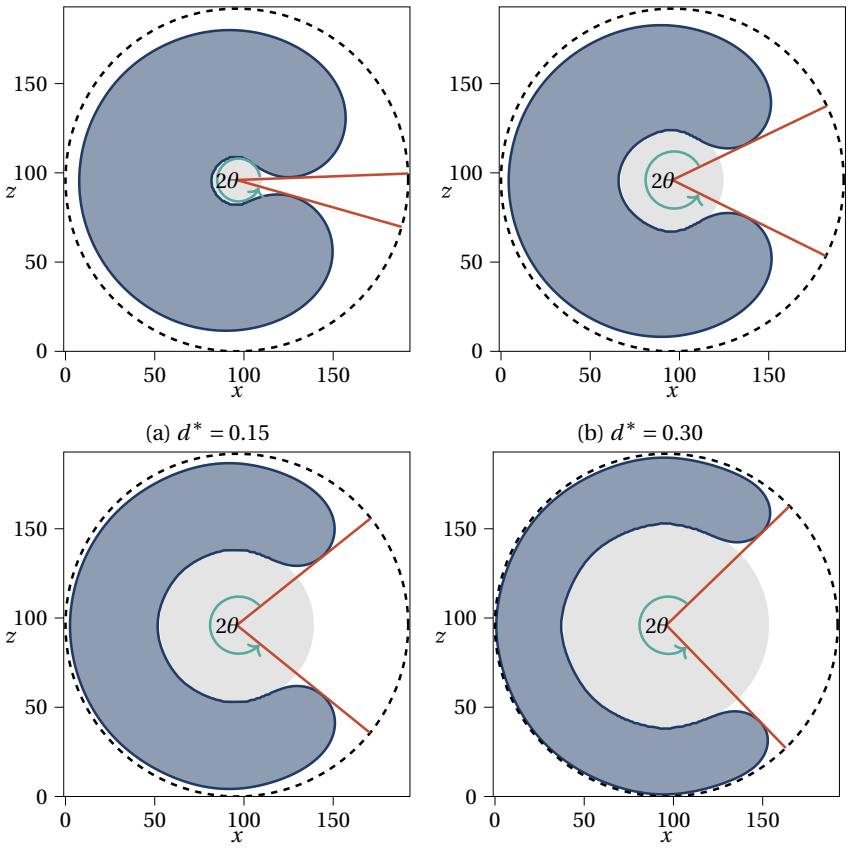
Figure 7.15: The effect of pipe ratio on the propagation of an annular Taylor bubble in a water-filled, annular conduit. The rise velocity is given in (a) and the angle of wrap in (b).

liquid film represents a significant portion of liquid drainage from above to below the Taylor bubble. The liquid film can be analysed under changing pipe ratios and varying fluid properties like viscosity or surface tension in future work. In this manner, valuable insight could be gathered to find closure relations between the diameter ratio and the size of the liquid bridge.

7.3 Conclusion

In this chapter, we analysed the behaviour of annular Taylor bubbles in annular pipe configurations. First, we performed a sensitivity analysis of the mobility parameter in the phase field formulation. It was found that the mobility parameter had a negligible impact ($\leq 1\%$) on the consistency of results for this case. However, values closer to 0.1 provided better stability. This case directly reproduces the results of previous work presented by Mitchell [79], in which a mobility parameter of $M = 0.05$ was used.

Following this, we examined the effect of pipe geometry by introducing eccentricity, which, to the best of the authors' knowledge, is not currently accounted for in velocity closure relations found in the literature. For the water/air-like case examined, only a small variation in velocity was observed. However, the angle of wrap of the Taylor bubble around the inner pipe changed significantly. Lastly, different sizes of the inner pipe were investigated to assess the impact of the pipe ratio on the rise of an annular Taylor bubble. These results were compared with the closure relation presented by Mitchell and Leonardi



(a) $d^* = 0.15$ (b) $d^* = 0.30$

(c) $d^* = 0.45$ (d) $d^* = 0.60$

Figure 7.16: Steady-state wrap angle, 2θ, for an annular Taylor bubble at a pipe ratio of (a) $d^* = 0.15$, (b) $d^* = 0.30$, (c) $d^* = 0.45$, (d) $d^* = 0.60$.

[118], formulated from a numerical study focusing on a pipe ratio of $d^* = 0.5$. The sizing of the inner pipe affects the dimensionless numbers on which this closure model depends, and it was found that this was sufficient to predict the rise velocities obtained in our simulations accurately.

CHAPTER 8

THERMOCAPILLARY FLOWS

Contents

In this chapter the conservative Allen-Cahn model is extended to simulate thermocapillary systems. After briefly introducing the thermocapillary systems analysed in this thesis in Section 8.1, we present a two-dimensional test scenario in Section 8.2. The goal is to verify the solution method and show its accuracy compared to existing literature. Building upon these results leads to the target simulations of this work, namely the analysis of droplets in a laser-heated microchannel. Numerical results for this scenario are first conducted in a two-dimensional setup in Section 8.3. From there, we extend the existing literature by conducting three-dimensional simulations in various setups. The simulations conducted in this work show that two-dimensional results have only limited meaning when extending to real three-dimensional setups. We present our conclusion in Section 8.5. The results shown here follow the author's contribution to the published article [5], published as part of the thesis.

8.1 Brief Introduction

The surrounding temperature influences the surface tension between the fluids in thermocapillary systems. This dynamic surface tension plays a driving role in microfluidic devices or reduced gravity environments. Thus, the temperature-dependent surface tension needs to be modelled carefully in these systems. From the application side, it becomes crucial to quantify the forces and to be able to control the motion of droplets and bubbles in the mentioned scenarios. In most cases, the dynamic surface tensions impose a shear force along the interface of the fluids, which leads to an inverse relation between the temperature. This inverse relation lets fluid migrate from hot regions to colder ones due to minimising the total surface tension [121, 201]. Furthermore, the effective manipulations of droplets have contributed to developing droplet-based microfluidic devices capable of rendering programmable and re-configurable operations [202, 203]. While droplets in microchannels can be controlled in numerous ways, e.g., using radiation pressure forces, it has been shown that the control with temperature manipulation is especially effective [121, 202, 204].

8.2 Planar Heated Channel

To simulate thermocapillary systems with the lattice Boltzmann method (LBM), necessary equations have been presented in Section 3.5. In this thesis, the model was implemented in WALBERLA and using LBMPY to provide an exceptionally flexible implementation. Our implementation is independent of the lattice stencil and compatible with single-relaxation-time (SRT), multiple-relaxation-time (MRT) or central-moment-based collision operators. In this chapter central-moment-based collision operators are used for all lattice Boltzmann (LB) steps and a D3Q19 lattice stencil for the phase field and hydrodynamic LB step (D2Q9 in two dimensions). For the thermal LB step, the choice of the lattice stencil is investigated, which is presented in the following.

This section validates the thermocapillary extension of the conservative Allen-Cahn model (CACM). This is done using the thermocapillary-driven motion of fluid in a heated microchannel. The test case is chosen for two reasons. First, it is possible to find an analytical solution for it, which is oftentimes impossible for thermocapillary flow problems, and second, it is a well-studied case by numerous authors [127, 128, 201].

The simulation setup is illustrated with Figure 8.1. Further, the domain is formed by a channel of length L and height H. Hence, the setup and its analytical solution given in Appendix A is two-dimensional. Nevertheless, Mitchell *et al.* [128] have used a pseudo-two-dimensional setup to validate their three-dimensional LB solver. This is done by extending the width of the channel by three lattice cells and applying periodic boundary conditions on the additional dimension. Due to the code generation approach our solver can be generated for two- or three-dimensional setups likewise. Thus, both variants are tested in this section. The initial phase field splits the domain into two sections of $\phi_L = 0$ and $\phi_H = 1$, corresponding to two fluids with their density ρ_L and ρ_H, respectively. At the western and eastern sides of the domain, periodic boundary conditions are used for all solvers. At the northern and southern sides of the domain, no-slip boundary conditions are used for the hydrodynamic and Allen-Cahn LB solver. For the temperature solver, on the other hand, Dirichlet boundary conditions are applied

$$T(x, -H/2) = T_h + T_0 \cos\left(\frac{2\pi}{L_\mathrm{c}} x\right), \tag{8.1}$$

$$T(x, H/2) = T_c, \tag{8.2}$$

where the temperatures are specified as $T_0 = 4$, $T_c = 10$, $T_h = 20$, and L_c is the characteristic length. The Dirichlet boundary conditions are realised with Equation (2.31) for LB temperature solver.

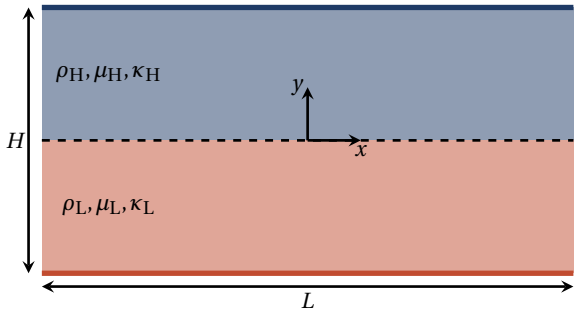


Figure 8.1: Schematic illustration of the two-dimensional heated microchannel used for validating the thermocapillary model.

To carry out the numerical simulation for the heated microchannel a characteristic length of $L_\mathrm{c} = 80$ lattice cells is defined. The characteristic length is equal to the height of the channel, and thus, the problem size is 160×80 cells. The fluid properties of both fluids are stated in Table 8.1. They result in the density ratio $\rho^* = 1$, the viscosity ratio $\mu^* = 1$, and the heat capacity ratio of $c_p^* = 1$. For the ratio of the thermal diffusivity, two test cases are constructed by choosing $\kappa_\mathrm{L} = 0.2$, and $\kappa_\mathrm{L} = 0.04$ corresponding to thermal diffusion ratios of $\kappa^* = 1$ and $\kappa^* = 1/5$, respectively. The resulting dimensionless parameters Ma, Re, and Ca are small ($\ll 1$), which means that the convective transport of momentum and energy can be neglected. As such, the interface in the simulation remains planar and is initialised as

$$\phi = \phi_0 + \phi_0 \tanh\left(\frac{y}{W/2}\right), \quad -\frac{H}{2} < y < \frac{H}{2}. \tag{8.3}$$

An analytic solution is provided for the simulation setup in Appendix A. With the analytic solution at hand, an error norm can be defined to measure the accuracy of the solver. In the scope of this thesis a L^2-norm is defined as,

$$L^2_{\text{norm}} = \sqrt{\frac{\sum_{i,j}(|\psi^{num}_{i,j}| - |\psi^{A}_{i,j}|)^2}{\sum_{i,j}|\psi^{A}_{i,j}|^2}}, \tag{8.4}$$

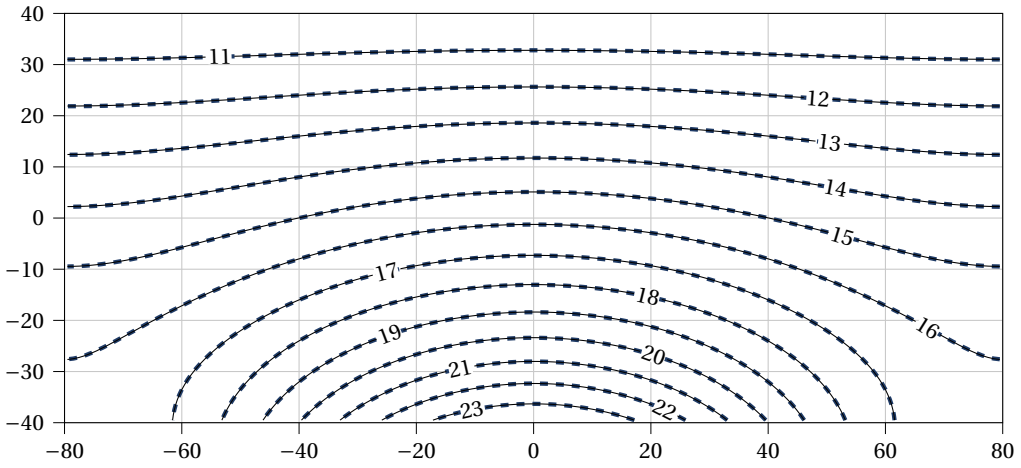
where ψ is a placeholder for temperature or velocity, respectively.

Table 8.1: Fluid properties including density, ρ, viscosity, μ, heat capacity, c_p, thermal diffusion, κ, reference surface tension, σ_{ref}, change of surface tension, σ_T, mobility, M, and the interface width, W, as well as the dimensionless parameters defined as the Marangoni, Ma, Reynolds, Re, and the Capillary, Ca number.

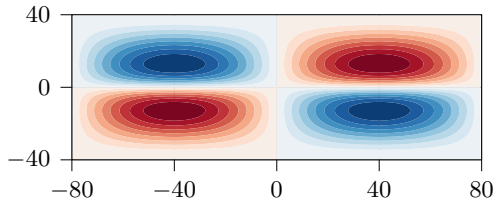
Parameter	Fluid H	Fluid L
ρ	1	1
μ	0.2	0.2
c_p	1	1
κ	0.2	0.2 (0.04)
σ_T	$-5\cdot10^{-4}$	
σ_{ref}	$2.5\cdot10^{-2}$	
M	$5\cdot10^{-2}$	
W	4	
	Formulation	Value
U_c	$\lvert\sigma_T\rvert/2\mu_H$	0.001 25
Ma	$\rho_H c_{p,H} L_c U/k_H$	0.1
Re	$\rho_H U L_c/\mu_H$	0.5
Ca	$U\mu_H/\sigma_{ref}$	0.01

A visual comparison of the analytical and the numerical solution of the temperature field is given in Figure 8.2 where the temperature contour lines are shown in Figure 8.2a. Dashed lines represent the analytical solution, while solid lines show the numerical solution. Furthermore, streamlines of the analytical (Figure 8.2b) and the numerical (Figure 8.2c) velocity field are illustrated in a side-by-side comparison. The error calculated with Equation (8.4) is stated in Table 8.2. The results of previous studies are added for comparison [126, 128, 205]. The study presented here applies different lattice stencils for the temperature LB solver. For the two-dimensional case, a D2Q9 stencil is used for all solvers. The pseudo-two-dimensional case on the other side is solved using a D3Q7, D3Q15, D3Q19, and a D3Q27 lattice stencil for the temperature solver and a D3Q19 stencil for the remaining solvers.

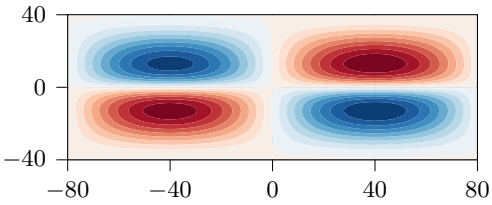
Our results are in good agreement with the literature. It can be seen that both the two-dimensional and pseudo-two-dimensional recover consistent agreement with the literature. The error of the temperature field is about one order of magnitude higher in the second configuration where $\kappa^* = 1/5$. Liu *et al.* [126] has given a possible explanation for this. The finite interface thickness of the phase-field model can not perfectly resolve the jump of the thermal diffusivity across the interface. It is important to note here that Liu *et al.* has used the Cahn-Hilliard equation for the interface tracking. Nevertheless, their conclusion also seems to be applicable for the CACM. Especially, the contour lines



(a) Temperature contours



(b) Analytical velocity streamlines



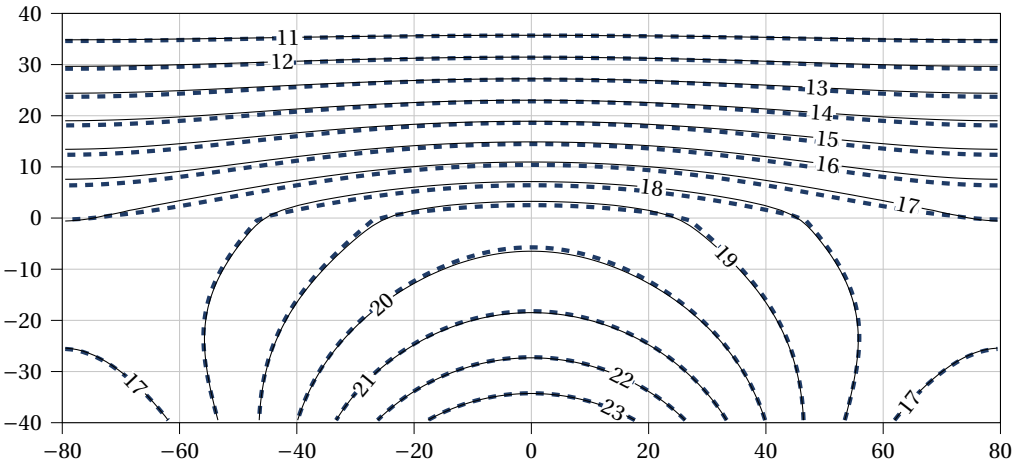
(c) Numerical velocity streamlines

Figure 8.2: Temperature contour lines (a) for thermocapillary driven layered flow with a thermal diffusivity ratio of $\kappa^* = 1$. Dashed contour lines indicate the analytical solution compared to the solid lines obtained through the presented simulation method. Also, the velocity streamlines of the analytical solution (b) and the numerically obtained velocity field (c) are shown.

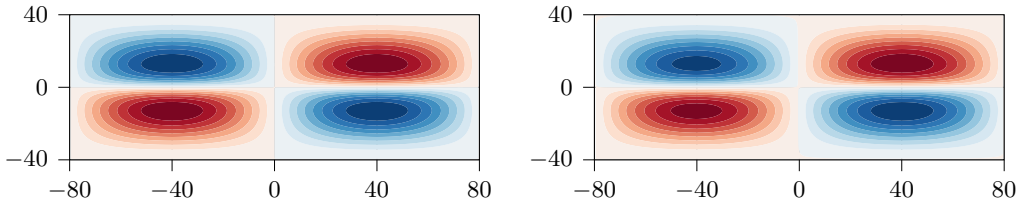
Table 8.2: Comparison of errors for the temperature and the velocity field after $4 \cdot 10^5$ timestep

L^2_{norm}	$\kappa^* = 1$		$\kappa^* = 1/5$	
Model setup	*Temperature*	*Velocity*	*Temperature*	*Velocity*
Present (D2Q9)	$1.98 \cdot 10^{-4}$	$13.27 \cdot 10^{-2}$	-	-
Present (D3Q7)	$1.92 \cdot 10^{-4}$	$14.42 \cdot 10^{-2}$	$5.25 \cdot 10^{-3}$	$17.51 \cdot 10^{-2}$
Present (D3Q15)	$1.93 \cdot 10^{-4}$	$14.42 \cdot 10^{-2}$	$5.25 \cdot 10^{-3}$	$17.51 \cdot 10^{-2}$
Present (D3Q19)	$1.93 \cdot 10^{-4}$	$14.42 \cdot 10^{-2}$	$5.25 \cdot 10^{-3}$	$17.51 \cdot 10^{-2}$
Present (D3Q27)	$1.93 \cdot 10^{-4}$	$14.42 \cdot 10^{-2}$	$5.25 \cdot 10^{-3}$	$17.51 \cdot 10^{-2}$
Mitchell *et al.* [2]	$8.27 \cdot 10^{-4}$	$11.83 \cdot 10^{-2}$	$7.65 \cdot 10^{-3}$	$14.18 \cdot 10^{-2}$
Liu *et al.* [126]	$2.25 \cdot 10^{-4}$	$5.71 \cdot 10^{-2}$	$5.23 \cdot 10^{-3}$	$8.41 \cdot 10^{-2}$
Majidi *et al.* [205]	$5.47 \cdot 10^{-4}$	$5.61 \cdot 10^{-2}$	$4.98 \cdot 10^{-3}$	$8.37 \cdot 10^{-2}$

of the temperature field given in Figure 8.3a show great consistency with their theory. At a low thermal diffusivity ratio, the isotherms become denser in the upper fluid, and the isotherms approaching the lower fluid tend to be normal at the interface. This indicates a heat transfer close to zero in the y-direction between the lower fluid and the interface.



(a) Temperature contours



(b) Analytical velocity streamlines

(c) Numerical velocity streamlines

Figure 8.3: Temperature contour lines (a) for thermocapillary driven layered flow with a thermal diffusivity ratio of $\kappa^* = 1/5$. Dashed contour lines indicate the analytical solution compared to the solid lines obtained through the presented simulation method. Also, the velocity streamlines of the analytical solution (b) and the numerically obtained velocity field (c) are shown.

For the pseudo-two-dimensional case, it is noticeable that no difference between the stencils was obtained. While this does not yet show the feasibility of complex real three-dimensional flow dynamics, it can already function as a guideline. Thus, using the D3Q7 stencil seems to be promising for simple thermocapillary flows, allowing minimal computational cost without significant impact on simulation results.

8.3 Droplet Motion with Local Laser Heating in 2D

Building on the results of the previous section, here, the model is applied to simulate the dynamics of a two-dimensional droplet in a laser-heated channel setup. This setup was previously analysed in the studies of Liu *et al.* [127]. Similarly to the work here, they used a phase-field LBM to study the development of the multiphase system, however, they have used the Cahn-Hillard equation to track the interface in contrast to our work which builds on the CACM. To approximate a laser-heat source in the channel, a simple Gaussian-like description is used,

$$q_T = \begin{cases} Q_s \exp\left(-2\frac{(\boldsymbol{x}-\boldsymbol{x_s})^2}{w_s^2}\right), & \text{if } \left[(\boldsymbol{x}-\boldsymbol{x_s})^2\right] \leq d_s^2 \\ 0, & \text{otherwise} \end{cases}, \tag{8.5}$$

where Q_s signifies the maximum heat flux generated by the laser, while $\boldsymbol{x_s}$ denotes the precise laser position. The z-component of the position vectors $\boldsymbol{x}$ and $\boldsymbol{x_s}$ was neglected in the two-dimensional setup. The remaining parameters of Equation (8.5) are defined as the heat dispersion d_s and the heat flux profile controlled by w_s. The simulation parameters of the whole setup, including the laser-heated source, are given in Table 8.3, and a schematic illustration of the setup is shown in Figure 8.4. Importantly, the parameters are chosen similarly to the study of Liu *et al.* [127], which allows a comparison between the simulations. The setup contains a semicircular droplet with radius $R = 32$ lattice cells in a rectangular channel with a size of $L_x \times L_y = 8R \times 2R$. Initially, the droplet is located at $\boldsymbol{x_c} = (2R+1, 0)$ with no initial velocity defined. To induce motion in the channel, a velocity of u_w is applied to the upper wall, while the lower wall remains stationary, resulting in a constant shear rate of $\gamma = {u_w}/{L_y}$.

Table 8.3: Fluid properties including density, ρ, viscosity, μ, heat capacity, c_p, thermal diffusion, κ, reference surface tension, σ_ref, change of surface tension, σ_T, mobility, M, and the interface width, W, as well as the dimensionless parameters defined as the Marangoni, Ma, Reynolds, Re, and the Capillary, Ca number.

Parameter	Fluid H	Fluid L
ρ	1	1
μ	0.2	0.2
c_p	1	1
κ	0.2	0.2
σ_T	$2 \cdot 10^{-4}$	
σ_ref	$5 \cdot 10^{-3}$	
M	$2 \cdot 10^{-3}$	
W	4	
Q_s	0.2	
d_s	8	
w_s	6	
	Formulation	Value
Ma	$\rho_H c_{p,H} LU / k_H$	0.08
Re	$\rho_H UL / \mu_H$	0.16
Ca	$U\mu_H / \sigma_{ref}$	0.01

The velocity is imposed to the hydrodynamic particle distribution function (PDF)s according to Equation (2.26), and no-slip boundary conditions are used for the stationary lower wall. For the phase-field LB step, no-slip boundary conditions are used for the upper and lower walls. In addition, temperature boundary conditions are set to zero ($T_\mathrm{ref} = 0$) for both the bottom and top walls. Periodic boundary conditions are used on the left and right walls for all three LB steps. Notably, the temperature of the fluid is solely influenced by a laser located at $\boldsymbol{x_s} = (181, 21)$.

The evolution of the two-dimensional droplet is investigated using various contact angles to the lower wall. The contact angle is imposed as explained in Section 3.4. The setup captures the droplet dynamics as it approaches the heated source, which creates an imbalance of surface tension forces. The droplet's hotter side exhibits lower surface tension. This imbalance of forces creates a shear force that opposes the droplet motion back towards the cooler region of the domain. With progressively lower three-phase

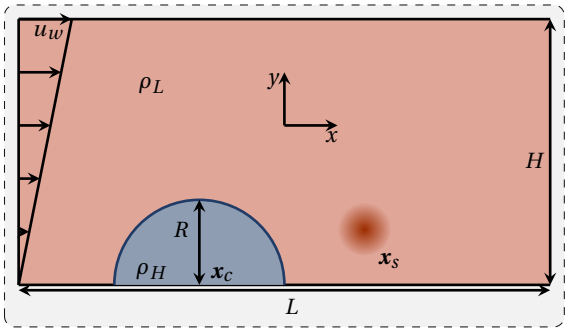


Figure 8.4: Schematic of the test domain used to simulate the thermocapillary-driven motion of a droplet in a channel.

contact angles, the droplet's equilibrium height is reduced, and the generated force is less aligned with the flow direction. As such, it is expected that the droplet will pass through a heated source for low contact angles that would otherwise trap a droplet with a higher contact angle.

The results of the numerical investigation are illustrated in Figure 8.5. Here, the temporal evolution of the x-component of the droplet's centre of mass is shown. The droplet dynamics are investigated for contact angles of $\alpha \in \{30°45°60°90°120°135°\}$. Our simulations are compared with the results of Liu *et al.* [127]. Simulations with a contact angle of $\alpha \in \{60°90°120°135°\}$ match very well with the results reported by Liu *et al.*. For higher contact angles, the shear flow acts stronger on the droplet, which increases its velocity. However, reaching the laser point blocks the droplet, and thus, it remains at rest. In the study of Liu *et al.*, droplets with a contact angle of $\alpha = 45°$ can bypass the laser point while these droplets are blocked in our simulation. In our studies, a contact angle of $\alpha = 30°$ is needed for the droplets to bypass the laser point. There could be various reasons for this, including the fact that the prior study used a Cahn-Hilliard approach to track the interface between the fluids, while here, the conservative Allen-Cahn equation (CACE) is employed. This also changes how the contact angle is enforced. To the author's knowledge, there is no additional literature available that studies this exact setup. Thus, future studies of this setup are necessary to obtain more insights into how the solution methodology influences the droplet motion. Nevertheless, the results presented here agree qualitatively with the results of Liu *et al.* [127]. This means droplets with higher contact angles are faster and can not bypass the laser point, while droplets with lower contact angles move slower and bypass the laser point when a certain contact angle α is applied.

8.4 Droplet Motion with Local Laser Heating in 3D

This section analyses the droplet motion in a three-dimensional laser-heated channel. This analysis expands the results of the previous section and the literature. The additional degree of freedom allows us to see motion perpendicular to the primary shear direction and increases complexity in the droplet-heated spot interaction. For this analysis, the same parameters as shown in Table 8.3 are used. The domain size was specified as $L_x \times L_y \times L_z = 16R \times 2R \times 8R$ to avoid boundary effects. A droplet was placed in the domain at rest at $\boldsymbol{x_c} = (2R+1, 0, 4R)$, and the droplet's radius was prescribed as $R = 32$ lattice cells. The location of the laser point is at $\boldsymbol{x_s} = (181, 21, 4R)$.

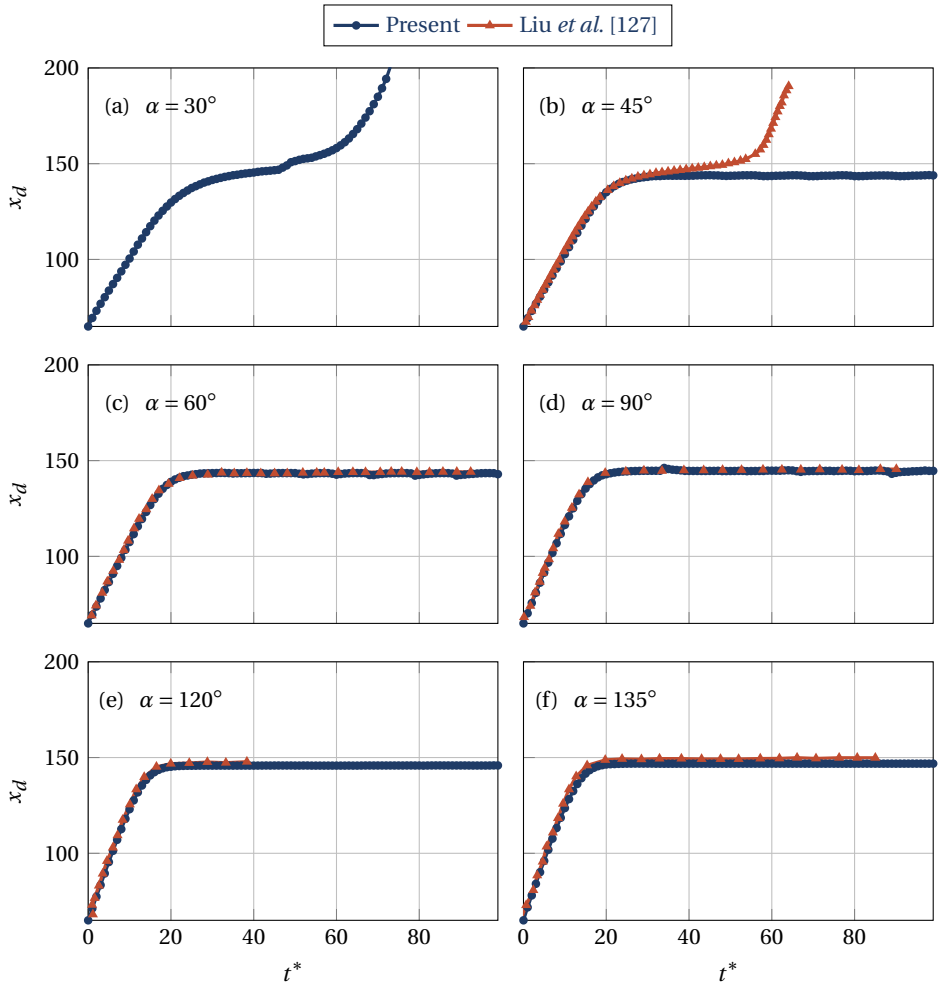


Figure 8.5: Droplet migration in 2D for different contact angles. The plots show the temporal evolution of the x-component of the droplet's centre of mass. When possible, a comparison was done with the results of Liu *et al.* [127]. The introduced heat source completely blocks droplets with a higher contact angle to the wall.

A three-dimensional view of the initial setup is shown in Figure 8.6a. At the start of the simulation, the temperature field is already evolved by applying t^* timesteps to the temperature LB solver exclusively. This is enough to obtain an evolved laser source as shown in Figure 8.6a. At this point, the velocity field, as well as the phase-field are still at rest. For better understanding, a schematic xy- and a xz-plane are presented in Figure 8.6c and Figure 8.6b.

Similar to Section 8.3, different contact angles are used to simulate the motion of the three-dimensional droplet. These contact angles were varied from 45° to 135°. The numerical simulation results are shown in Figure 8.7. Here, an exemplary temporal evolution of a droplet with $\alpha = 90°$ is shown in Figure 8.7a. Furthermore, Figure 8.7b shows the displacement in the x-plane of various droplets with different contact angles, while the x-component of their velocity is shown in Figure 8.7c. Unlike the two-dimensional simulation, all droplets can pass the laser point, and none can be stopped. Here, the two-dimensional planar analogue does not replicate the impact of a spherical heat source on the semi-spherical droplet in a three-dimensional domain. The velocity profile of the

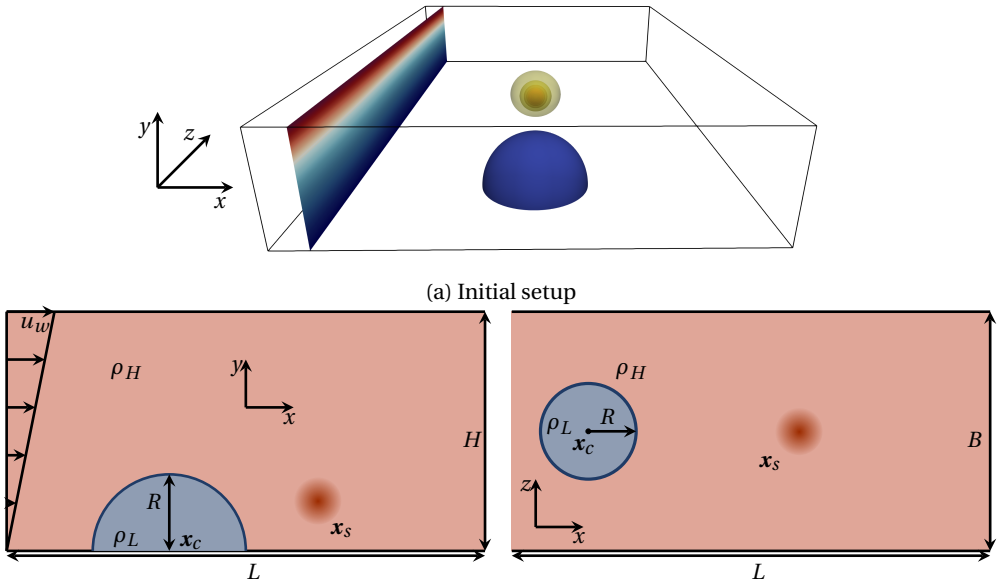
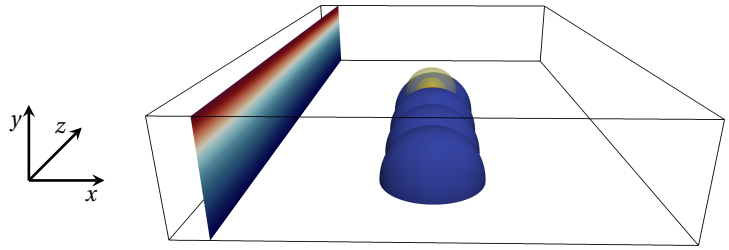


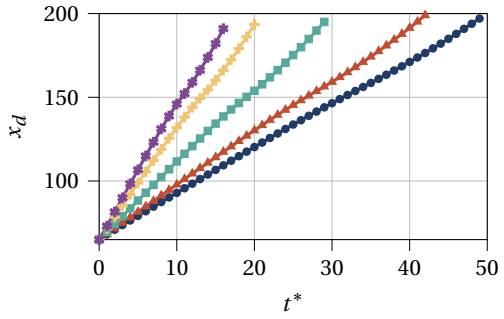
Figure 8.6: Configuration of the test domain for the three-dimensional extension of the simulation of a thermocapillary droplet, showing (a) the initial setup of the droplet in a shear flow channel, (b) an elevation of the domain in the xy-plane, and (c) a plan view of the domain in the xz-plane. A single heat source is introduced to the thermal solver, similar to the analysis in Section 8.3.

droplets also confirms this behaviour. At first, all droplets are accelerated due to the shear flow (this acceleration is higher for droplets with higher contact angles). When they reach the proximity of the laser point, the velocity decreases. However, the laser point in this simulation is not strong enough to block the droplets. Instead, they bypass the laser point, which results in an acceleration once the laser point is behind the droplets. Once the droplets are far enough from the laser source, their velocity reaches a steady state similar to that of the initial phase.

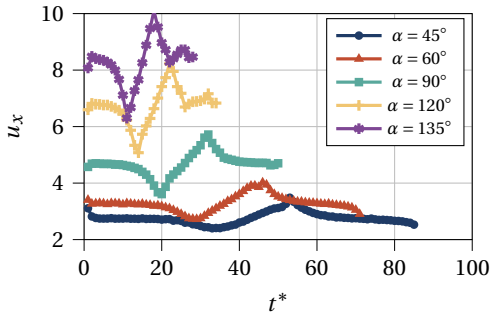
The test case was modified as a next step by increasing the heat flux by a factor of five to $Q_s = 1$. This was done to investigate if it is possible to block droplets just by increasing the strength of the laser source. The results of the second simulation are shown in Figure 8.8. An exemplary temporal evolution of a droplet with $\alpha = 90°$ is shown in Figure 8.8a. Furthermore, Figure 8.8b illustrates the displacement in the x-plane of various droplets with different contact angles, while their respective z-plane is given in Figure 8.8c. It can be observed that the droplets move towards the laser point until $\approx t^* = 10$. All droplets seem to stop at this time, which looks similar to the blocking behaviour shown in Section 8.3. However, a displacement on the respective z-plane of the droplets can be observed at the same point. This indicates that the droplets now move in the additional direction available in the three-dimensional simulation. The described situation is pictured clearly in Figure 8.8c, where the z-coordinate of the centroid of the droplets is plotted as a function of the dimensionless time t^*. Interestingly, this was the case for all droplets. However, it was found that droplets with lower contact angles needed more time and tended to move around in a lower range than droplets with higher contact angles. To obtain more insight, a small parameter study with heat fluxes $Q_s \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ is shown in Figure 8.10. This indicates configurations in which



(a) Temporal motion of a droplet with contact angle $\alpha = 90°$ in a three-dimensional shear flow



(b) x-coordinate of the centroid of the droplets



(c) x-component of the velocity of the droplets

Figure 8.7: Droplet migration in a three-dimensional channel flow with various contact angles, showing (a) the evolution of a droplet with 90° contact angle where each droplet in the figure represents the movement during a period of $t^* = 10$ and (b) the x-coordinate of the centroid of droplets with various contact angles as a function of the dimensionless time on the left and its respective velocity on the right (c). A maximal heat flux of $Q_s = 0.2$ was applied for the laser point.

droplets with lower contact angles can pass through the heat source without displacement in the z-direction, while droplets with higher contact angles move around the heat source in the z plane. Most prominently, this situation can be observed for $Q_s = 0.6$. In this configuration, some droplets can bypass the laser point directly while other droplets ($\alpha > 60°$) bypass the laser point in the z-direction.

Finally, a flow configuration with two laser points was studied. The two heat sources were shifted by $\pm 1/2 R$ in the z-direction in this setup. This means the laser points are located at $\boldsymbol{x_{s,1}} = (181, 21, 3.5R)$ and $\boldsymbol{x_{s,2}} = (181, 21, 4.5R)$ respectively. The three-dimensional evolution of a droplet with contact angle $\alpha = 90°$ can be seen in Figure 8.9a. The trajectory of the droplets in the x- and z-directions are shown in Figure 8.9b and Figure 8.9c, respectively. In this case, the blocking of the droplets can be observed for a longer time. However, with the applied heat flux magnitude, the droplets with high contact angles migrate transversely and pass both laser points. Furthermore, the droplet with contact angle $\alpha = 45°$ passes through the two heat sources. It thus shows very similar behaviour to that observed in the two-dimensional case, where droplets with low contact angle could slip underneath the laser point. These test cases provide an understanding of the migratory behaviour of a droplet in shear and showcase how the introduced modelling capability can be applied to design effective droplet capture and or manipulation devices. It is clearly shown that a three-dimensional setup shows a larger spectrum of behaviours.

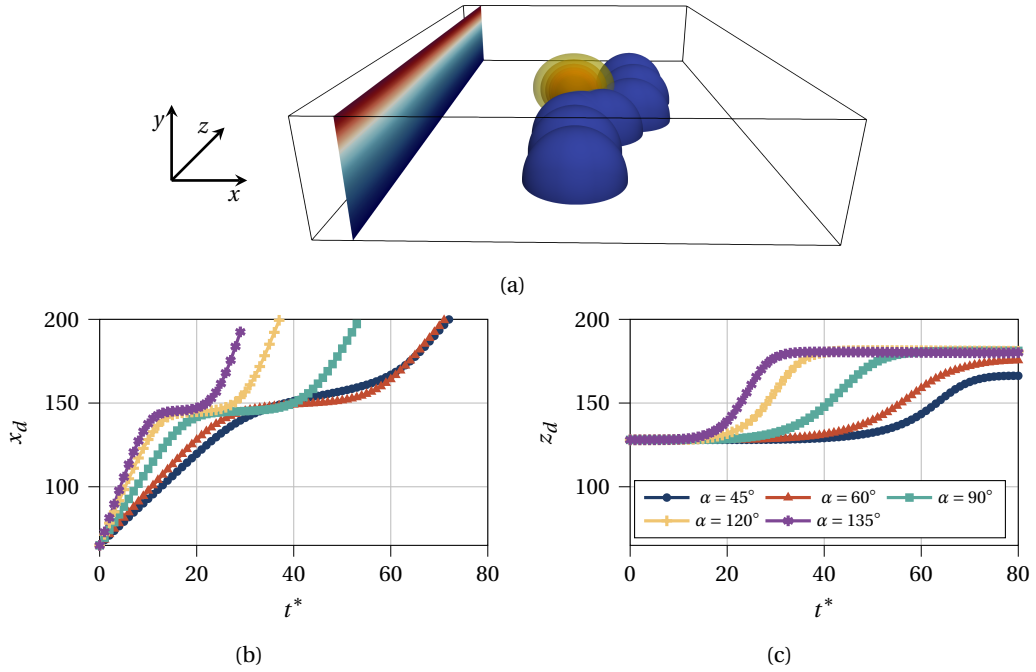


Figure 8.8: Droplet migration in a three-dimensional channel flow with various contact angles showing (a) the evolution of a droplet with 90° contact angle where each droplet in the figure represents the movement during a period of $t^* = 10$ and (b) displacement of the droplets in the x- and (c) z-directions with various contact angles. A maximal heat flux of $Q_s = 1$ was used for the laser point.

The impact of increasing the heat flux Q_s is shown in Figure 8.10. Droplets with contact angles of $\alpha \in \{45°, 60°, 90°, 120°, 135°\}$ move in a channel with a single heat source. At a heat flux of $Q_s = 0.2$, the droplets pass through the laser-heated point without blockage. However, with increasing heat flux, it remains impossible to show droplet blocking for a different reason. This can be best understood by looking at the z-component of the centroid of the droplets z_d, which is pictured in the figures of the second column. As the heat flux Q_s increases, droplets move around the heat source in the z-direction. The transition can be observed best at $Q_s = 0.6$ where droplets with high contact angle move around the heat source while droplets with low contact angle pass underneath it. In the hope of observing droplet blockage, additional experiments have been conducted using two laser points. In these experiments, the laser points were shifted by $\pm 1/2 R$ and $\pm 2/3 R$ in the z-direction respectively. The first results of the first experiment where ($\boldsymbol{x_{s,1}} = (181, 21, 3.5R)$ and $\boldsymbol{x_{s,2}} = (181, 21, 4.5R)$) are shown in Figure 8.11. In this case, it is possible to show droplet blockage for simulations with higher heat flux ($Q_s \geq 0.6$). However, a secondary effect can be noticed. The heat sources are rather close to each other, and thus, droplets with higher contact angles are still able to pass by on the z-direction. This can be seen especially well for $Q_s = 1$ where droplets with contact angle $\alpha > 90°$ pass rather quickly, while droplets with lower contact angle stay blocked for a while until they slowly move in the z-direction. In the last experiment, the heat sources are shifted further apart by $\pm 2/3 R$. This means the laser points are located at $\boldsymbol{x_{s,1}} = (181, 21, (4 - 2/3)R)$ and $\boldsymbol{x_{s,2}} = (181, 21, (4 + 2/3)R)$ respectively. The results for this experiment are pictured in

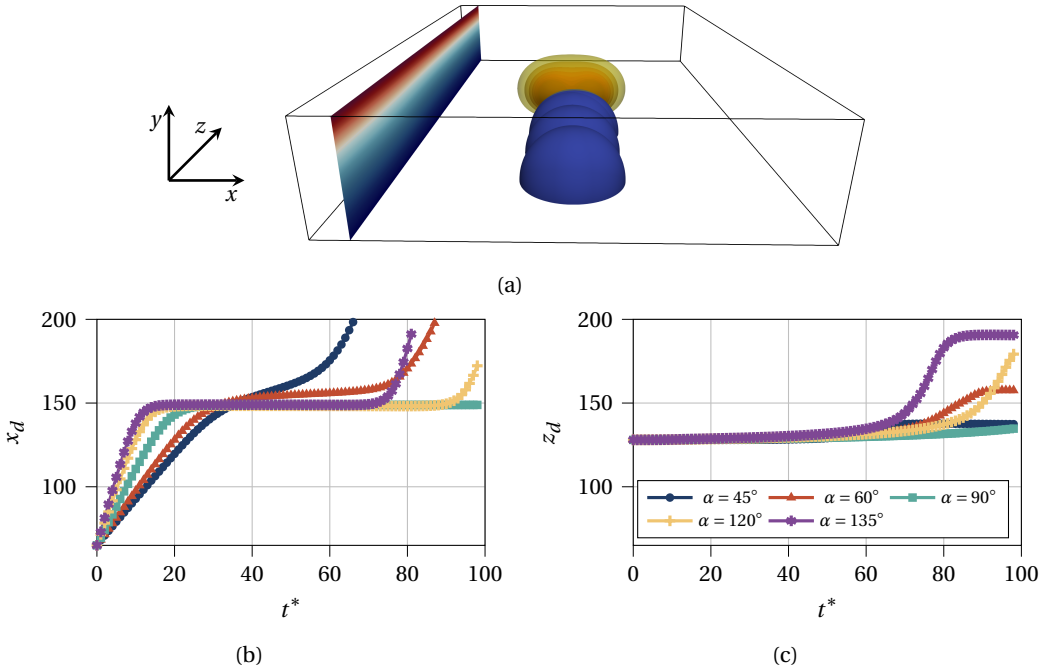


Figure 8.9: Droplet migration in a three-dimensional channel flow with various contact angles showing (a) the evolution of a droplet with 90° contact angle where each droplet in the figure represents the movement during a period of $t^* = 10$ and (b) displacement of the droplets in the x- and (c) z-directions with various contact angles. Two laser sources are shifted by $\pm 1/2R$. Their heat flux is $Q_s = 0.6$.

Figure 8.12. Again, with lower heat flux ($Q_s \leq 0.6$), all droplets slip through the laser points. However, in none of the configurations, droplets move around the heat sources in the z-direction. With $Q_s = 1$, all the analysed droplets show blocking throughout the complete simulation time.

8.5 Conclusion

In this chapter, we presented results obtained using a newly implemented thermocapillary model in WALBERLA. The energy equation in this model was solved using a third LB solver. After comparing various lattice stencils in an academic test case, we concluded that the D3Q7 stencil is sufficiently accurate for solving the energy equation.

This model and its implementation were applied to study the potential of local heating (e.g. a source analogous to laser tweezers) to stop the shear migration of droplets. It was found that a two-dimensional representation of this inherently three-dimensional scenario was inadequate for determining when capture would occur. In the three-dimensional test case, using only a single heated source showed that droplets circumnavigated the hot spot, allowing migration to continue. However, when two heated sources were applied, the droplet could be stopped, though the placement of the heat sources and the droplet's contact angle had a significant influence on the outcome. The model and implementation presented here are expected to enable future research into the design of microfluidic devices that rely on droplet manipulation and control.

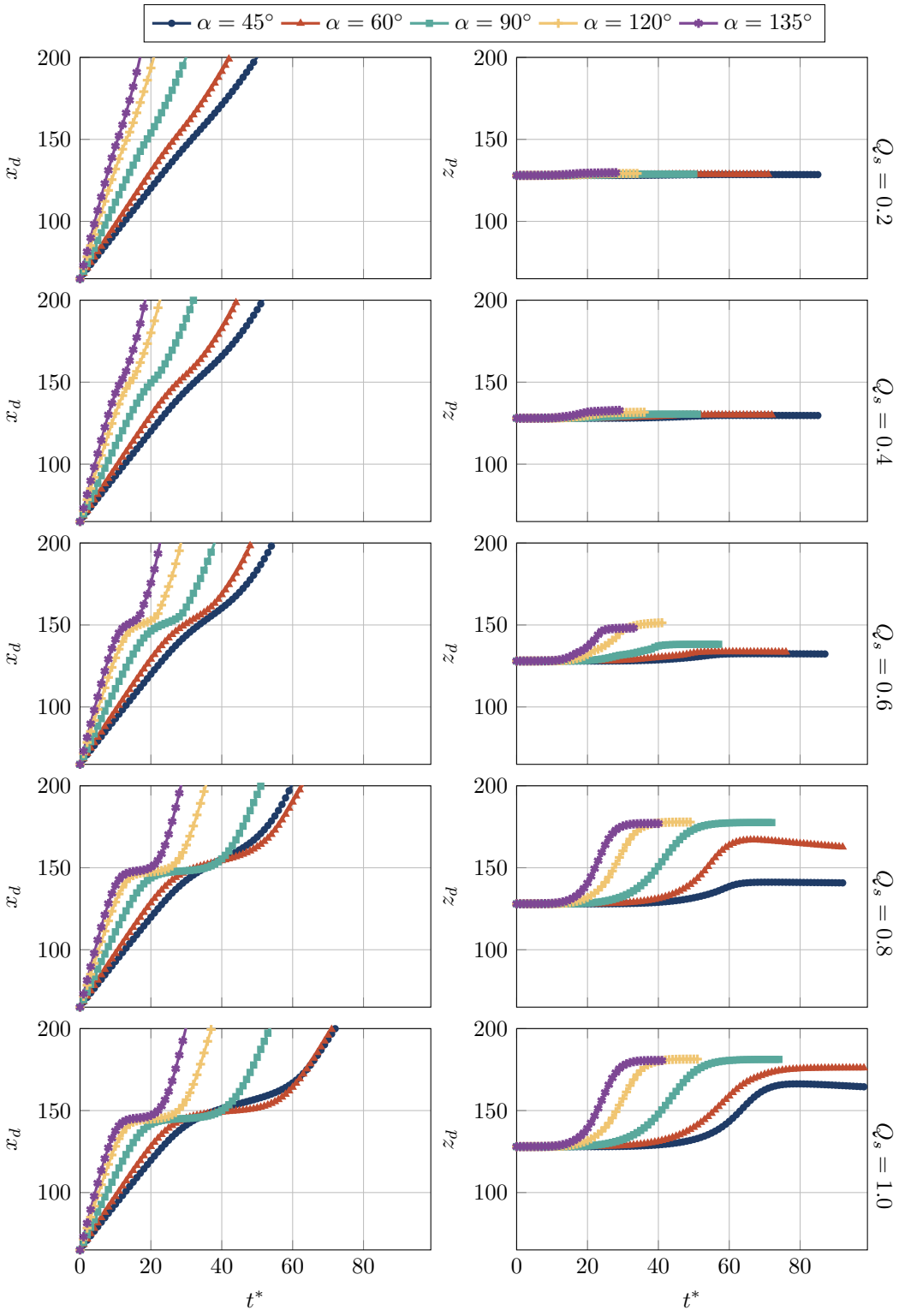


Figure 8.10: Droplet migration in a three-dimensional channel flow with various contact angles with a single laser point. The figures on the left column show the displacement of the droplets in the xy-plane, while the right column shows the displacement in the xz-plane. The respective heat flux is given on the right of each row.

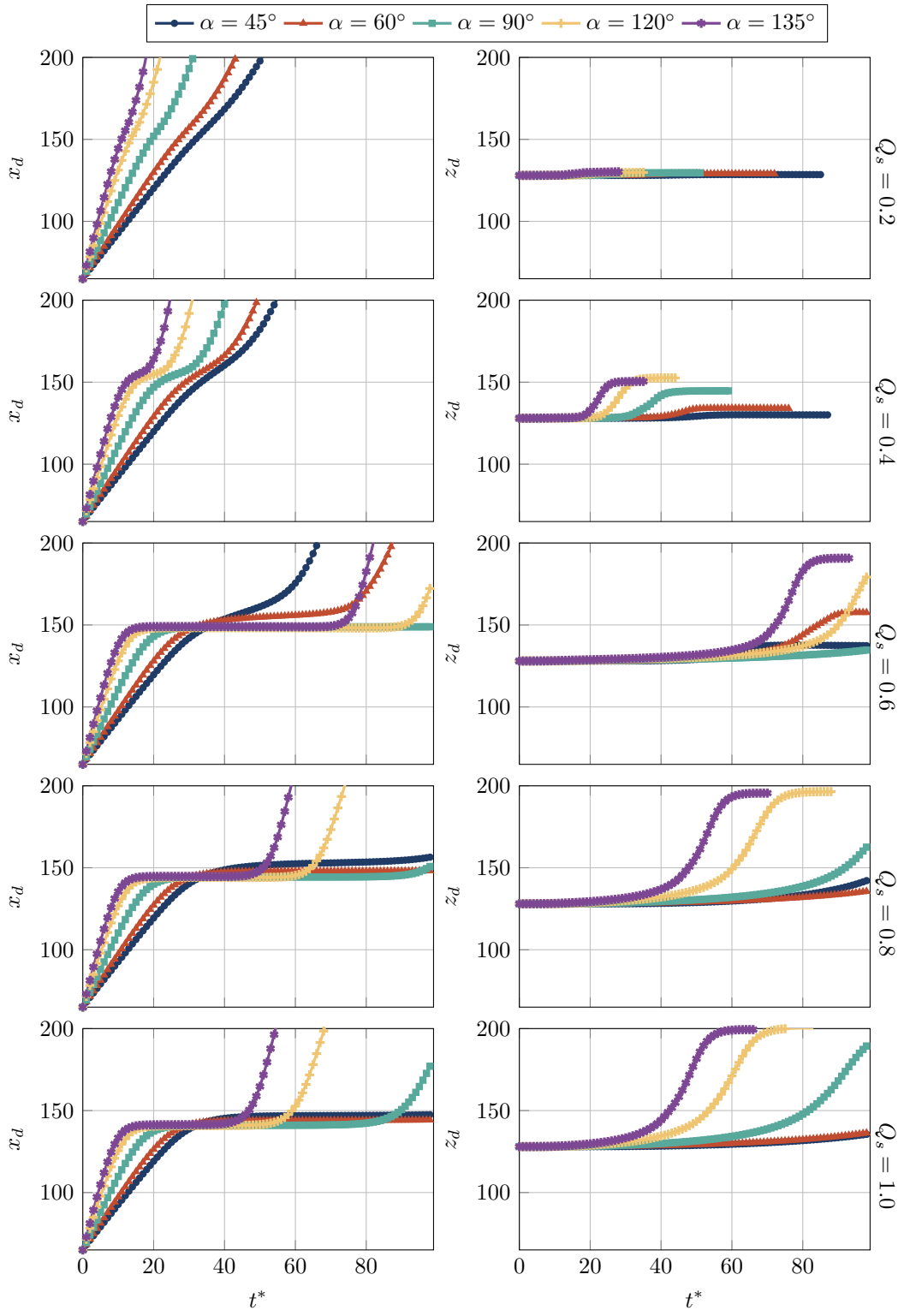


Figure 8.11: Droplet migration in a three-dimensional channel flow with various contact angles and two single laser point. The two laser points are shifted by $\pm 1/2R$ in the z-position. The figures on the left column show the displacement of the droplets in the xy-plane, while the right column shows the displacement in the xz-plane. The respective heat flux is given on the right of each row.

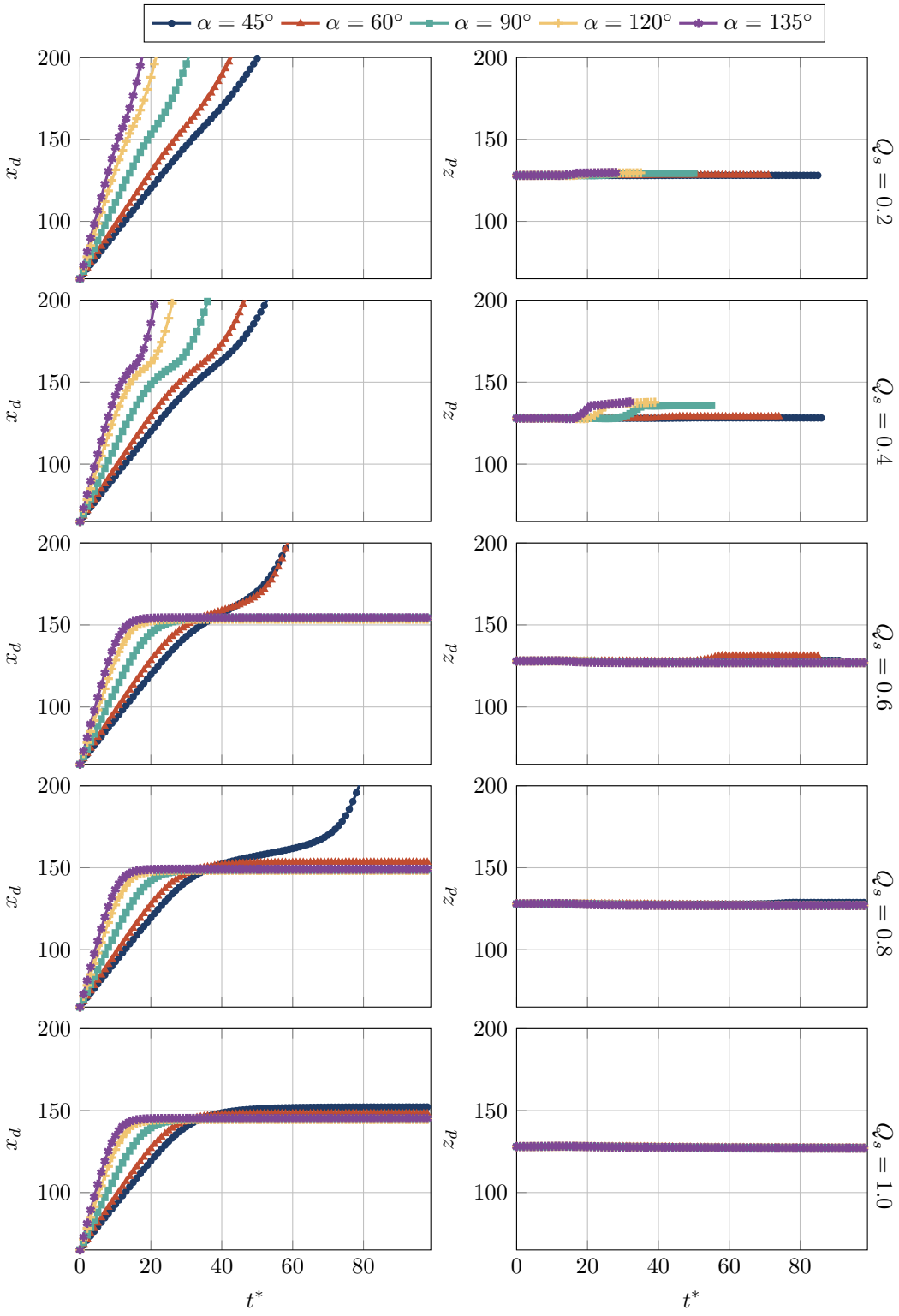


Figure 8.12: Droplet migration in a three-dimensional channel flow with various contact angles and two single laser point. The two laser points are shifted by $\pm 3/3R$ in the z-position. The figures on the left column show the displacement of the droplets in the xy-plane, while the right column shows the displacement in the xz-plane. The respective heat flux is given on the right of each row.

CHAPTER 9

PERFORMANCE RESULTS

Contents

In this chapter, we present the performance of the resulting compute kernels for the lattice Boltzmann method. This performance analysis is done on several state-of-the-art computing systems detailed in Section 9.1. To properly report benchmark results, metrics need to be defined to ensure the results' comparability. We introduce the most relevant metrics in Section 9.2. Afterwards, we explain the benchmark setups for the different evaluations in detail. Before we look at large-scale benchmarks, we present single-node analysis in various architectures in Section 9.4. This is done to highlight the generality of the code generation approach and to have a basis for further scaling benchmarks. Finally, we present large-scale performance benchmarks and discuss uniform (see Section 9.5) and non-uniform mesh configurations (see Section 9.6). We finish the chapter by analysing further performance increases in Section 9.7, that can be achieved when lower precision data formats are used.

9.1 Computing Systems

The performance analysis for this thesis is done on three supercomputing systems. These are the SuperMUC-NG [1], the JUWELS Booster [2], and the LUMI [3] computing system. At the time of writing, these systems hold place 50, 21 and 5 of the TOP500 list [4] (effective June 2024). In the following sections, we give details on their respective hardware, followed by details about the software stack we used on the respective systems.

9.1.1 Hardware Configuration

The hardware configuration of the SuperMUC-NG supercomputer is summarised in Table 9.1. The x86-based Intel processor provides all computing capacity. This processor supports 512-bit Advanced Vector Extensions (AVX512) single instruction, multiple data (SIMD) instructions, which can be generated by PYSTENCILS. The SuperMUC-NG supercomputer consists of 6336 so-called *thin* nodes, which are relevant to our analysis and constitute the largest part of the cluster. In total, the system can achieve a peak performance of $26.3 \cdot 10^{15}$ floating point operations per second (FLOPS) using its 304 128 central processing unit (CPU) cores. For this thesis, we had access to 3072 compute nodes, which is approximately half the cluster size. At this scale, we were required to utilise at least four islands. An island is an organisational unit consisting of 792 compute nodes, with faster network connections within each island.

The hardware configuration of the JUWELS Booster system is summarised in Table 9.2, with a technical overview provided by Kesselheim *et al.* [206]. The computing power is primarily based on the NVIDIA A100 general purpose graphics processing unit (GPGPU), and four of them are implemented per compute node. This thesis focuses exclusively on the accelerator component of the JUWELS Booster system; therefore, the CPU of the compute nodes is not discussed. The JUWELS Booster system comprises a total of 936 nodes, resulting in nearly 4000 NVIDIA A100 GPGPUs. With this setup, a total performance of $78.98 \cdot 10^{15}$ FLOPS can be achieved. For the purposes of this work, we had access to 256 compute nodes, which provided 1024 GPGPUs.

[1] https://doku.lrz.de/hardware-of-supermuc-ng-phase-1-11482553.html
[2] https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html
[3] https://docs.lumi-supercomputer.eu/hardware/lumig/
[4] https://www.top500.org/lists/top500/list/2024/06/

Table 9.1: Hardware configuration of the thin nodes of the SuperMUC-NG supercomputer

processor	Intel Skylake Xeon Platinum 8174
sockets per node	2
cores per processor	24
processor base frequency	3.1 GHz
memory per node	96 GB
nodes per island	792
number of islands	8
total number of nodes	6336
total number of cores	304 128
total memory	608 TB
interconnect	OmniPath network with 100 Gbit s^{-1}
network within island	fat tree
connection between islands	pruned 1:4
theoretical peak performance	$26.3 \cdot 10^{15}$ FLOPS

A detailed overview of the hardware configuration of the LUMI-G partition of the LUMI supercomputer is provided in Table 9.2. The LUMI-G partition derives its computing power primarily from the accelerators installed in each node. Each compute node is equipped with four AMD MI250X GPGPUs, each of which contains two Graphics Compute Dies (GCDs), effectively dividing the accelerator into two separate units. Since the GCDs have their own memory and computing chip, two message passing interface (MPI) ranks are required to utilise a single GPGPU. The two GCDs are connected via the in-package Infinity Fabric interface, with a theoretical bidirectional bandwidth of up to 400 GB s^{-1}. In this work, only complete GPGPUs are compared; that is, an AMD MI250X GPGPU is always utilised using two MPI ranks. The LUMI-G partition consists of 2978 compute nodes, providing almost 12 000 AMD MI250X GPGPUs, which delivers a total peak performance of $531.51 \cdot 10^{15}$ FLOPS. As a result, the system surpasses half an exaFLOP and is considered one of the largest pre-exascale systems. For this thesis, we had access to 1024 compute nodes, equating to 4096 AMD MI250X GPGPUs.

Table 9.2: Hardware configuration of the JUWELS Booster supercomputer and the GPGPU-partition of the LUMI supercomputer

	JUWELS Booster	LUMI-G
accelerator	NVIDIA A100	AMD MI250X
GCDs	-	2
accelerator per node	4	4
accelerator memory per node	160 GB	512 GB
total number of nodes	936	2978
total number of accelerators	3744	11 912
network topology	dragonfly	dragonfly
nodes per group	48	124
groups	20	24
theoretical peak performance	$78.98 \cdot 10^{15}$ FLOPS	$531.51 \cdot 10^{15}$ FLOPS

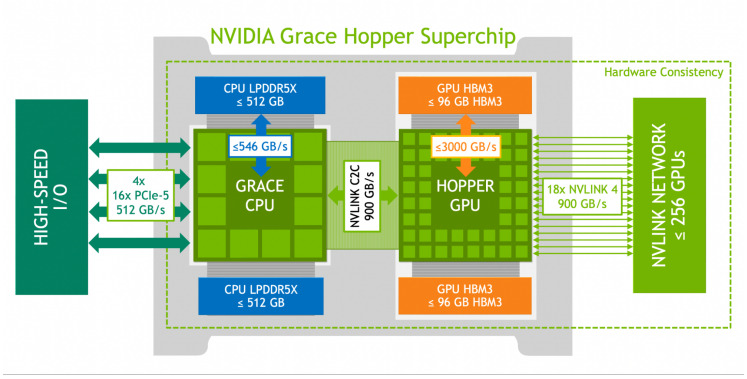


Figure 9.1: Hardware overview of NVIDIA Grace Hopper superchip [208].

Lastly, the performance of the NVIDIA Grace Hopper superchip, which is installed in the Calypso cluster of CERFACS[5], is analysed. This chip will form the basis of Europe's first exascale supercomputer called Jupiter [207]. Thus, the accelerator will play a crucial role in future applications. An overview image of the GH200 chip is shown in Figure 9.1. In contrast to previous generations of accelerators, the chip also hosts an ARM-based CPU, connected with NVLINK C2C to the H100 GPGPU. The Grace CPU supports vector instructions in the form of the Neon [6] and the Scalable Vector Extension (SVE) [7] instruction set. Neon intrinsics are for 128 bit-wide registers. On the other hand, the SVE instruction set is a vector extension of the A64 instruction set of the Armv8-A architecture. The significant difference between the SVE and the Neon instruction set is that the size of the registers is not static but ranges from a minimum of 128 bits up to a maximum of 2048 in 128-bit wide units. In the following performance analysis, both the CPU and GPGPU will be analysed.

9.1.2 Software Stack

To compile WALBERLA on SuperMUC-NG, the Intel *oneAPI* compiler with version 2021.4.0 is used with the highest optimisations (i.e., `-O3` and `-xhost` but without fast math). This means the compiler is motivated to introduce SIMD instructions automatically if not already done by the code generator. Intel-MPI 2019 is used for inter-node communication. All simulations shown in this work are executed without using the energy-aware runtime of the SuperMUC-NG cluster by setting `-ear=off`. This benchmark setting is recommended, ensuring a constant CPU clock frequency. As shown in Table 9.1 792, compute nodes form a single island, and inter-island communication is slower than intra-island communication. For this reason, the option `switches` is used to force the jobs to use the lowest number of islands possible. Again, this setting is recommended to ensure the reproducibility of the benchmarks.

On the JUWELS Booster system, the software stack of *Stage 2024* is used. Software stages are logical bundles that provide software versions that are compatible with each other. Within this software stage, compute unified device architecture (CUDA) version 12.2 and GCC-compiler version 12.3 are used to compile the code. Again, the highest compiler

[5] https://cerfacs.fr/en/cerfacs-computer-resources/
[6] https://developer.arm.com/Architectures/Neon
[7] https://developer.arm.com/documentation/102476/latest/

optimisations are used without fast math for every benchmark. For inter-node communication, OpenMPI version 4.1.5 is linked against WALBERLA. Importantly, the MPI package used here is build CUDA-aware. This means that pointers to device memory are directly understood by MPI; thus, copies to host memory are unnecessary within WALBERLA. Furthermore, fast direct connections between GPGPUs can reduce the communication overhead during runtime. The four GPGPUs within a single node are connected with NVLink3 [8] which can provide a databridge with a speed of about 600 GB s^{-1} between the GPGPUs.

The software on the LUMI supercomputer is organised into software stacks. Similar to the stages on the JUWELS Booster system, these form a logical purpose of providing compatible software versions within a stack. In this work, the *LUMI/23.09* stage is used together with the LUMI-G partition module and the *lumi-CPEtools/1.1-cpeCray-23.09* module. This module provides a Clang compiler [9] with version 14.0.0 is provided. The Clang version on the LUMI-G partition utilises the *HIPCC* [10] compiler driver utility, which is a wrapper around the Clang compiler to enable target compilation of AMD GPGPU code which uses the *HIP* language. Furthermore, the Cray software stack provides an MPI library with MPI standard 3.1, which is linked to the AMD *ROCm* framework to enable direct communication between connected GPGPUs. Thus, the MPI library understands device pointers directly, and a manual copy to the host memory is obsolete. The GCDs on different AMD MI250X accelerators are directly connected with either a single or double Infinity Fabric link, which can provide a peak bidirectional bandwidth of 100 GB s^{-1} and 200 GB s^{-1}, respectively.

9.2 Performance Modelling and Metrics

A simple metric to report a simulation's performance is the time-to-solution. A big advantage of this metric is that it is simple to measure. However, it hides certain crucial aspects when comparing the performance of different simulation runs. Most importantly, this concerns the used resources. For example, a simulation that took only half of the time as another simulation but used four times the hardware resources should not be considered as more performant. Thus, a suitable performance metric should take the resource usage and the simulation time into account. A common metric used for the lattice Boltzmann method (LBM) is lattice updates per second (LUPS), which is defined as

$$\text{LUPS} = \frac{\text{total number of simulation steps} * \text{total number of cells}}{\text{total simulation time}}. \tag{9.1}$$

Essentially, LUPS reports how many cells can be updated in a certain amount of time. While this metric considers the number of cells, it does not consider the hardware resources. Therefore, this chapter will report LUPS per hardware unit. Defining a suitable hardware unit can be complex as well. In this work, a hardware unit is defined as a full chip. This means that a single node of SuperMUC-NG should be defined with two hardware units as two Intel chips are used, while a single node of JUWELS Booster or LUMI-G

[8] https://www.nvidia.com/en-us/data-center/nvlink/
[9] https://clang.llvm.org/get_started.html
[10] https://rocm.docs.amd.com/projects/HIPCC/en/latest/

should be defined as four hardware units due to its four NVIDIA A100 or AMD MI250X GPGPUs respectively. Modern CPU and GPGPU systems can typically update more than 10^9 lattice cells per second. Therefore, all performance results will be reported in giga lattice updates per second (GLUPS).

Due to the intensive memory consumption of the LBM, it is commonly reported that the stream-collide compute kernel is limited by the memory bandwidth [26]. Therefore, the theoretical peak performance can be estimated by a roofline analysis as

$$P_{\max} = \frac{b_{\mathrm{s}}}{n_{\mathrm{b}}} \tag{9.2}$$

where b_{s} describes the maximum available memory bandwidth and n_{b} the number of bytes that need to be transferred per lattice cell. The roofline model sets an upper bound to the performance of a programme [209]. The underlying assumption is that data transfers through the memory hierarchy overlap with code execution on the cores. While the roofline model can not describe complex phenomena inside the chip's memory hierarchy, it is typically enough for the stream-collide compute kernel, which typically only accesses direct neighbour cells. More complex cases are covered, e.g., by the Execution-Cache-Memory (ECM) model, which considers the full memory hierarchy [210].

To determine the maximal memory bandwidth $b_{\mathrm{s,max}}$ of the Intel Xeon Platinum 8174 installed on SuperMUC-NG and the Grace CPU, the LIKWID performance tools are used [211]. In particular, LIKWID provides bandwidth benchmarks with the tool `likwid-bench`. On both CPUs, the benchmark *update* is executed on the full chip, i.e., using all 24 cores of the Intel Xeon Platinum 8174 and 72 cores of the Grace CPU respectively. The *update* benchmark loops through a single array and updates each element with itself ($A[i] = A[i]$). Therefore, it has a load/store ratio of one. This benchmark is chosen because it best resembles the behaviour of highly optimised lattice Boltzmann (LB) compute kernels using in-place streaming patterns as shown by Wittmann *et al.* [172]. The resulting maximal memory bandwidth can be found in Table 9.3.

Table 9.3: Maximum memory bandwidth of all architectures analysed in this work. The maximum bandwidth $b_{\mathrm{s,max}}$ on the CPUs is determined using the *update* benchmark of `likwid-bench`, while a handwritten *update* kernel was used on GPGPUs.

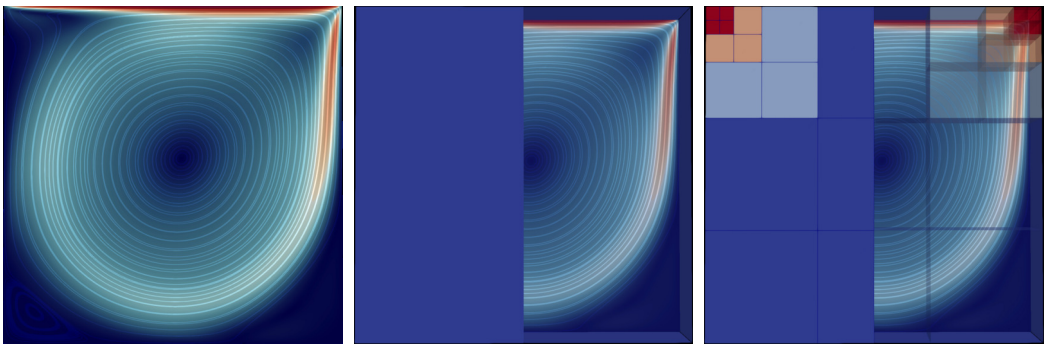
Architecture	$b_{\mathrm{s,max}}$
Intel Xeon Platinum 8174, 3.1 GHz, 24 cores	94 GB s^{-1}
NVIDIA A100 (40 GB)	1376 GB s^{-1}
AMD MI250X (128 GB)	2476 GB s^{-1}
NVIDIA H100 (96 GB)	3666 GB s^{-1}
Grace CPU, 3.1 GHz, 72 cores	272 GB s^{-1}

Since GPGPUs are not covered by the benchmarks contained in `likwid-bench`, a handwritten update kernel is executed. The resulting maximum bandwidth $b_{\mathrm{s,max}}$ is reported in Table 9.3.

9.3 Benchmark Setups

The following sections systematically analyse the performance of the generated LB kernels. This is done in three steps: The first step analyses the single-chip performance (see Section 9.4). A good single-chip performance is crucial for several reasons. Since the performance of a single chip sets the starting point for any large-scale problems, it is a critical first target for optimisations. Starting with a suboptimal single-chip performance can lead to wrong conclusions from scaling experiments since the communication overhead in large problems is more hidden by the longer time it takes to execute the suboptimal compute kernel. Therefore, it can lead to an over-optimistic scaling result. Furthermore, the optimised single-chip performance forms a good baseline for efforts to decrease the energy consumption of the programme by decreasing the frequency of any underused component of a chip [212]. Lastly, analysing the single-chip performance will give a baseline performance that can be used to put large-scale runs into perspective.

A synthetic benchmark is used to evaluate the performance of the parallel LBM algorithm on uniform and nonuniform grids. This synthetic benchmark aims to solve the lid-driven cavity problem. The problem is defined on a unit box $\Omega = [0,1] \times [0,1] \times [0,1]$ with velocity bounce back boundary conditions at $y = 1$ and resting walls in every other direction. An overview of the simulation setup is shown in Figure 9.2.



(a) xy-plane (b) Uniform setup (c) Non-Uniform setup

Figure 9.2: Setup of the lid-driven cavity performance benchmark. In (a), the xy-plane of the developed flow is shown. The simulation consists of moving wall on the top, while resting walls are used for every other wall. High velocities are coloured in red, while low velocities are shown in blue. In (b), the uniform domain decomposition is shown using one block per process and in (c), the non-uniform domain decomposition is shown where the top corners are refined. Blocks on the coarsest level are shown in blue, and the finest blocks are shown in red.

One block per process is allocated to resolve the problem on a uniform domain (see Figure 9.2b). This setup is analysed in two common scaling scenarios. The first one aims to analyse the weak scalability. Here, the scalability where the problem size is increased with the number of MPI ranks. In this way, every MPI rank always keeps the same workload. This setup is achieved by distributing more blocks according to the number of ranks and, thus, increasing the resolution of the problem accordingly. The second scaling analysis aims to analyse the strong scalability. This means the problem size is kept constant, and only the number of MPI ranks is increased. In this thesis, this is achieved by defining a total domain size and increasing the number of blocks while decreasing their size, e.g., using fewer cells per block.

Analysing the parallel performance on a uniform domain forms the basis for analysing the performance on a non-uniform domain. The setup to resolve the lid-driven cavity on a non-uniform grid is shown in Figure 9.2c. For this case, four mesh levels are used, and the edges on the top of the domain are resolved with the finest mesh size. This setup is identical to the work of Florian Schornbaum [29]. This means the weak scaling setup is distributed from 16 MPI ranks onwards to have between 6 and 7 blocks on each MPI rank. Thus, every rank holds four blocks of the finest resolution, one or two blocks of level two, one or no block of level one and one or no block of level zero. The properties of this setup are given in Table 9.4. To increase the problem size with the number of MPI ranks, the domain is elongated in the z-direction.

On the other hand, the domain size is fixed for the analysis of the strong scalability, and every MPI rank holds exactly one fine block and either one or no block from the other levels. In this way, each MPI rank ends up with one or two blocks. With more MPI ranks, the blocks will hold fewer and fewer cells, which results in more blocks overall while keeping the domain size constant.

Table 9.4: Memory requirements and workload of the lid-driven cavity problem setup shown in Figure 9.2c. Since cells of finer resolution are updated more frequently, the workload strongly increases on finer mesh levels. In this setup, more than 80% of the workload is on the finest mesh level, which only covers about 1.4% of the total domain.

	$L=0$	$L=1$	$L=2$	$L=3$
domain coverage ratio	77.78%	16.67%	4.17%	1.39%
workload share	1.10%	3.76%	15.05%	80.13%
memory share	6.54%	11.22%	22.43%	59.81%

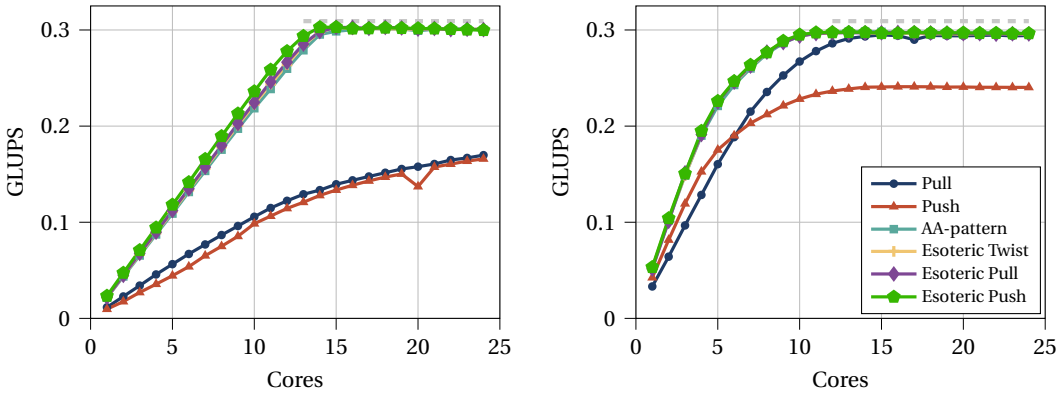
The total performance is reported in GLUPS for the single-chip benchmarks. The performance is divided by the number of chips used for all scaling benchmarks on the other side. This means for the CPU runs on SuperMUC-NG the performance is given per Intel Xeon Platinum 8174 processor, while on JUWELS Booster and LUMI-G the performance is given per NVIDIA A100 and AMD MI250X GPGPU respectively.

9.4 Single Node Performance

To analyse the single chip performance on the Intel Xeon Platinum 8174 processor and the Grace CPU, only the collide kernel of the LBM is executed. We chose these chips to analyse one x86 and an ARM-based architecture. This means boundary conditions and the data exchange between WALBERLA's blocks are neglected. All benchmarks are performed MPI-only with a single block of 128^3 cells per MPI rank. On both architectures, first, the simplest collision model is selected, i.e., the single-relaxation-time (SRT) collision model with a D3Q19 lattice stencil. The resulting compute kernel is generated with all streaming patterns that are supported by LBMPY. These are the pull and the push streaming pattern, which use two particle distribution function (PDF) arrays, and the AA-pattern, the Esoteric Twist, the Esoteric Pull and the Esoteric Push streaming pattern, which operate on a single PDF array. The idea of this benchmark is to analyse the behaviour of the different memory access patterns first. Afterwards, the complexity of the compute kernel is changed by switching from a D3Q19 stencil to a D3Q27 stencil and analysing various collision models. These are the SRT, multiple-relaxation-time (MRT), regularised central moment (R-C),

regularised cumulant (R-K) and the cumulant (K17) collision operator. The idea of the second benchmark is to analyse the performance under an increasing computational intensity as the collision models introduce more and more operations. The arithmetic operations of each kernel are shown in Table 4.2.

In Figure 9.3, the resulting single-chip performance on the Intel Xeon Platinum 8174 processor is shown for different streaming patterns. In Figure 9.3a, the in-place streaming patterns saturate the memory bandwidth at about 14 CPU cores and reach the maximum achievable performance of 0.31 GLUPS. A dashed grey line indicates the theoretical peak performance, calculated using the measured maximum bandwidth in Table 9.3. Both two-field streaming patterns can not saturate the memory bandwidth and lose about 40 % of performance.



(a) Non-vectorised

(b) Vectorised

Figure 9.3: Comparision of different streaming patterns on a single Intel Xeon Platinum 8174 processor on SuperMUC-NG. The kernels are generated using (a) no SIMD instructions and (b) using AVX512 SIMD instructions. All kernels are compiled with the highest compiler optimisations. This means in (a) the compiler is motivated to introduce vector instructions automatically. The dashed grey line indicates the maximum performance.

Using AVX512 SIMD instructions changes the picture, as shown in Figure 9.3b. Now, the pull streaming pattern can also saturate the memory bandwidth. The reason for this is the usage of non-temporal stores. This means the stored values on the temporary PDF vector are not automatically reloaded to the processor cache but only stored in the main memory. PySTENCILS can generate SIMD instructions which enforce this behaviour. However, non-temporal store instructions can only be applied to an aligned memory address. In the code generation process, PySTENCILS analyses the abstract syntax tree (AST) and is able to introduce temporary pointer locations on the outer loops in a way that only aligned memory addresses occur in the inner-most loop. Since the pull streaming pattern stores all values cell local for every store operation, a non-temporal AVX512 intrinsic can be used. The push streaming pattern writes to neighbours and performs unaligned stores, which non-temporal store instructions can not enhance on x86 architectures. Therefore, the resulting compute kernel cannot use the available memory bandwidth equally efficiently.

For the in-place streaming patterns, non-temporal store operations play no role since the PDF array is updated in place. Nevertheless, enhancing the compute kernels with AVX512 instructions results in an earlier saturation, which means that the kernels can utilise the full memory bandwidth already with ten CPU cores. It is thus a non-negligible optimisation, which can save energy. Furthermore, the results presented here reveal that all inplace streaming patterns show very similar behaviour on the Intel Xeon Platinum 8174 processor.

As a next step, compute kernels are generated with a D3Q27 stencil, the Esoteric Twist streaming pattern, SIMD instructions and various collision models. The resulting performance is shown in Figure 9.4. Due to the extended memory consumption of the D3Q27 stencil, the maximum performance (indicated as a grey dashed line) is decreased. Nevertheless, every collision model can saturate the maximum bandwidth at around 11 CPU cores. Except for the K17 model which needs a slightly higher core count for the saturation. Nevertheless, this is an important insight as it shows that most complex collision models, like the cumulant K17 model, can be used without causing any performance overhead compared to other collision models using the same discrete velocity set.

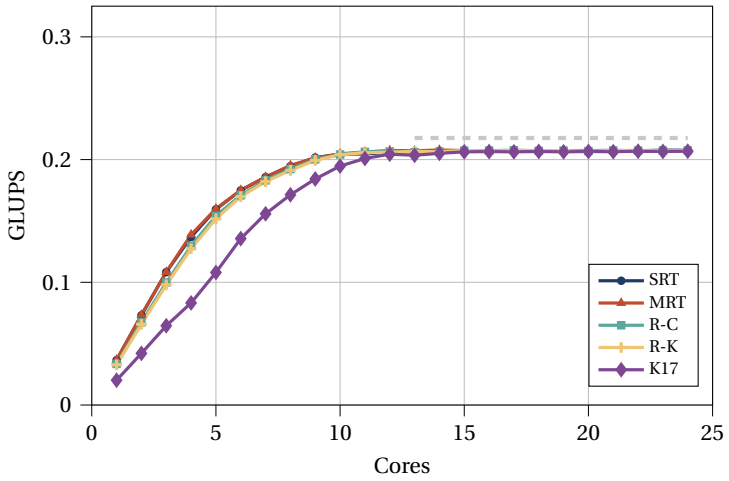
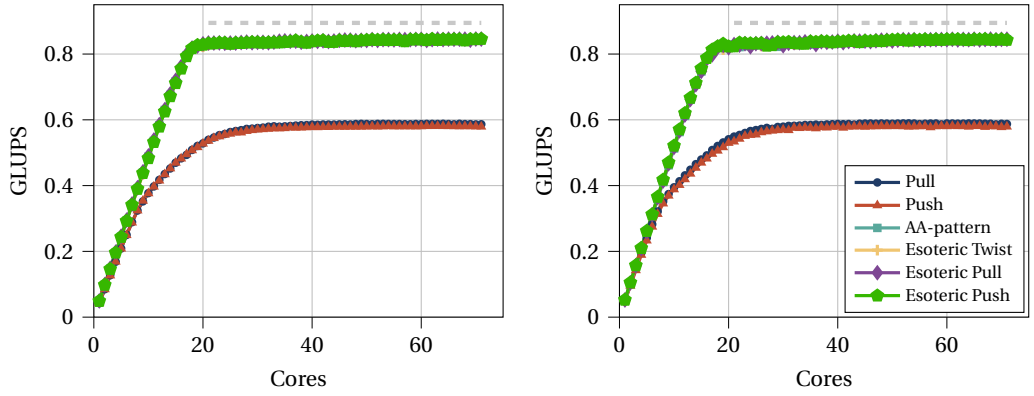


Figure 9.4: Comparision of different collision models on a single Intel Xeon Platinum 8174 processor on SuperMUC-NG. The kernels are generated using the D3Q27 velocity set, AVX512 SIMD instructions and the Esoteric Twist streaming pattern. All kernels are compiled with the highest compiler optimisations, and the grey dashed line indicates the theoretical peak performance.

Following the results of the x86 architecture, the same performance analysis is performed on the Grace CPU. As a first step, the influence of the streaming step is investigated by using an SRT collision model with the D3Q19 velocity set. The results of this investigation are shown in Figure 9.5, where the generated compute kernels use no SIMD instruction in Figure 9.5a and SVE instructions in Figure 9.5b. Similar to the results on the Intel Xeon Platinum 8174 processor, it is possible to saturate the maximum memory bandwidth within a single chip. This is possible by using an in-place streaming pattern. The pull or push streaming pattern can not saturate the memory bandwidth and eventually misses about 25 % of the maximum performance.

To improve this situation, non-temporal stores within the SVE instruction set are generated in the compute kernels of Figure 9.5b. However, this shows no noticeable effect, and the resulting picture is similar. It must be noted that PYSTENCILS supports all SIMD instruction sets that can be executed on the Grace CPU. Therefore, Neon SIMD

instructions were also benchmarked, which showed no noticeable difference. This is expected since the Grace CPU operates on 128-bit wide registers, which means that in this case, there is no difference between SVE and Neon. SVE offers more specific SIMD instructions like scatter and gather operations. However, these play no role in the experiment here.



(a) Non-vectorised

(b) Vectorised

Figure 9.5: Comparision of different streaming patterns on a single Grace CPU. The kernels are generated using (a) no SIMD instructions and (b) using SVE SIMD instructions. All kernels are compiled with the highest compiler optimisations. Hence, the compiler is motivated to introduce intrinsics automatically in (a). The dashed grey line indicates the maximum performance.

Analysing the effect of the collision model is shown in Figure 9.6. Again, all collision models can saturate the memory bandwidth of the CPU. On the Grace CPU, the minimal needed cores to saturate the bandwidth varies more, which manifests in almost 40 cores needed to saturate the K17 collision model. Nevertheless, even the most complex collision model shows a comparable single node performance on 72 cores as the simplest collision model.

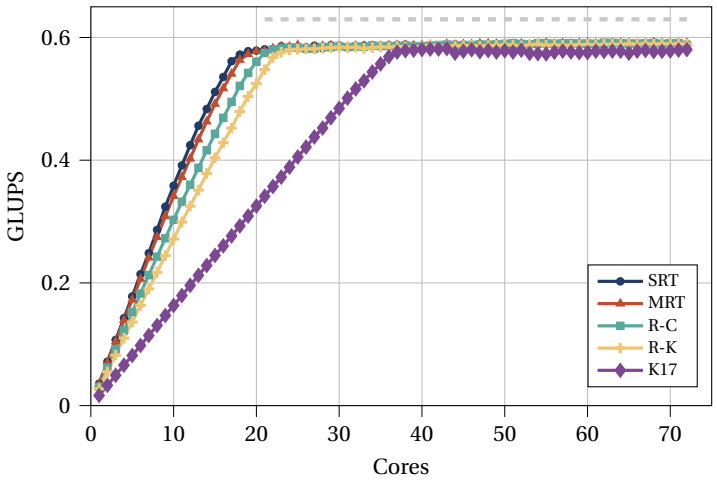


Figure 9.6: Comparision of different collision models on a single Grace CPU. The kernels are generated using the D3Q27 velocity set and the Esoteric Twist streaming pattern. SIMD instructions are not utilised since they seem to show no noticeable difference in the generated compute kernels in Figure 9.5. All kernels are compiled with the highest compiler optimisations, and the grey dashed line indicates the theoretical peak performance.

The analysis of different chip architectures is extended to GPGPUs as a next step. Here, the same systematic analysis is performed by first looking at the streaming pattern shown in Figure 9.7. The stream-collide compute kernel is executed on a square domain with 320^3 cells per GPGPU. Comparing the different streaming patterns on the NVIDIA A100, H100, and the AMD MI250X shows that the pull streaming pattern generally delivers the best absolute performance. This is because the pull streaming pattern uses only aligned writes by writing all Q PDF values at the centre of the cell. The opposite of the pull streaming pattern is the push streaming pattern, which writes all $Q-1$ PDFs to the neighbouring cell. It can be seen that this streaming pattern generally shows the worst performance. All in-place streaming patterns analysed in the scope of this thesis show a performance close to the pull streaming pattern since they employ a mixture of aligned and unaligned writes. Lehmann previously reported a similar result [40]. Furthermore, in his study, he emphasised the advantage of streaming patterns from the Esoteric family since they implicitly perform a noslip boundary condition. However, it must be noted that it only works for the simple noslip boundary condition without interpolation.

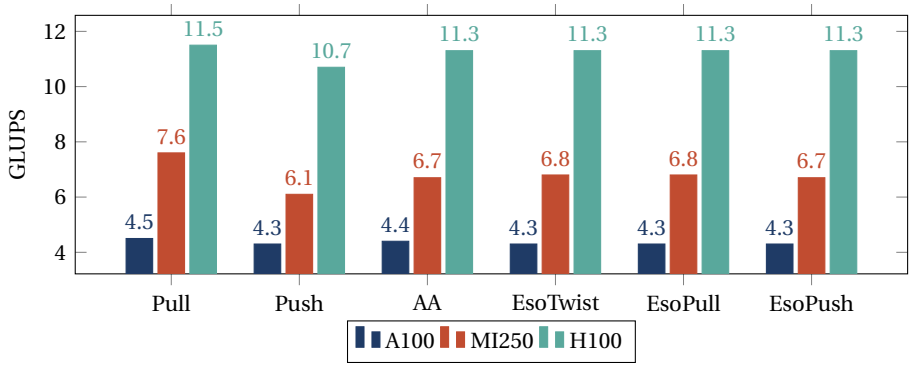


Figure 9.7: Comparision of different streaming patterns on an NVIDIA A100, AMD MI250X and an NVIDIA H100 GPGPU.

The results of using the Esoteric Twist streaming pattern with various collision models are shown in Figure 9.8. Besides the reduced performance due to the extended velocity set, it can be shown clearly that all collision models show similar absolute performance. Therefore, it can be concluded that the sophisticated mathematical optimisation within the PYSTENCILS framework pays off fully. Even the most complex K17 cumulant kernel can be optimised to stay memory-bound.

9.5 Scaling Behaviour on Uniform Mesh Configurations

Building upon the insights gained from the single-chip performance benchmarks, the scaling behaviour of the lid-driven cavity is analysed using a uniform grid resolution. Unlike the previous analysis, this investigation incorporates boundary conditions and data exchange within WALBERLA's block-structured framework. Each process is assigned a single block in all scenarios, meaning that data exchanges occur only via non-blocking MPI routines. The benchmarks focus on the scaling performance of the SRT collision model using a D3Q19 lattice stencil, with the Esoteric Twist streaming pattern applied across all compute kernels.

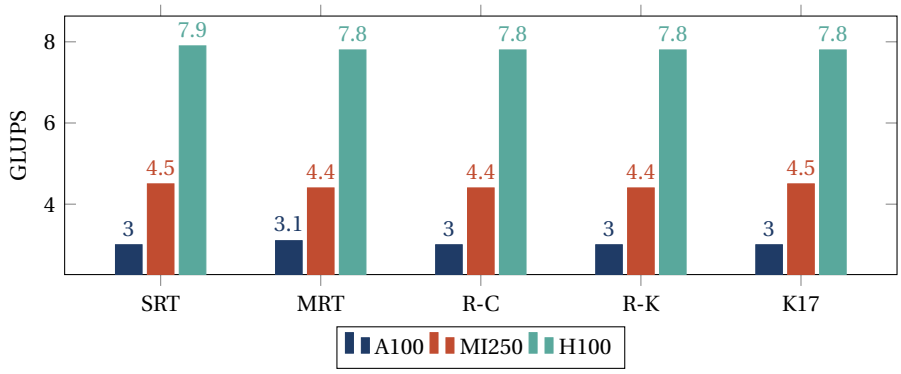


Figure 9.8: Comparision of different collision models on an NVIDIA A100, AMD MI250X and an NVIDIA H100 GPGPU.

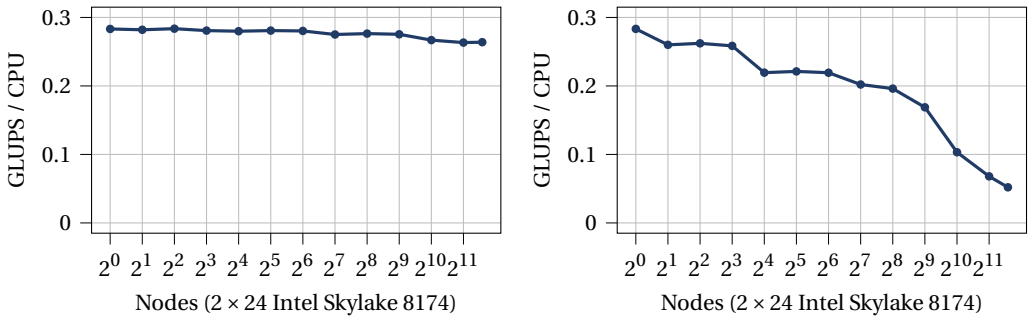
As previously discussed, the choice of collision model often has minimal impact on runtime performance, as the LBM is often bound by the hardware's bandwidth. Nevertheless, for the sake of completeness, an extension to the benchmarks is provided in Appendix B. There, we include results for the regularised cumulant collision operator with a D3Q19 stencil and the K17 model with its D3Q27 stencil alongside the SRT model presented here. These models were selected due to their increased computational complexity compared to the simple SRT model.

9.5.1 SuperMUC-NG

The weak and strong scaling results on the SuperMUC-NG supercomputer are presented in Figure 9.9. For the weak scaling benchmark (compare Figure 9.9a), $4.2 \cdot 10^6$ cells are allocated per core, and the simulation is scaled from a single node to 3072 compute nodes, comprising 147 456 CPU cores. This benchmark demonstrates near-perfect scalability from 48 to 147 456 cores. At the maximum scale, 147 456 cores are used to simulate a domain containing $6.18 \cdot 10^{11}$ cells, achieving a total performance of nearly $1.6 \cdot 10^3$ GLUPS.

It is worth noting that the performance shows a slight decline from 1024 nodes onwards. This decrease coincides with using more than one island of SuperMUC-NG, resulting in increased communication overhead due to the slower inter-island connection. Despite this, the overhead is almost entirely mitigated, allowing the benchmark to maintain good scalability even at the largest scales.

The strong scaling benchmark (compare Figure 9.9b) was conducted using a total domain size of $147.1 \cdot 10^6$ cells. As the number of MPI ranks increases, the subdomains become progressively smaller, leading to a reduction in performance per core. At 1024 nodes, the strong scalability drops below 50 %, which is anticipated due to the increased communication overhead caused by the slower network connection between two islands of SuperMUC-NG. Despite this, 2464 timesteps per second can still be achieved on 3072 compute nodes. At this scale, each core processes fewer than 1000 lattice cells.

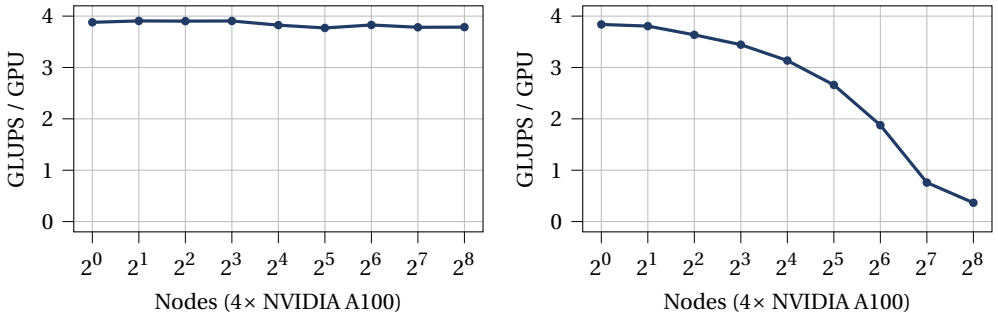


(a) Weak scaling (b) Strong scaling

Figure 9.9: Weak (a) and strong (b) scaling behaviour of the lid-driven cavity problem on SuperMUC-NG using a uniform mesh resolution.

9.5.2 JUWELS Booster

To analyse the weak scaling behaviour on JUWELS Booster, a domain size of $134 \cdot 10^6$ cells is allocated to each GPGPU. As the number of GPGPUs increases, the domain is scaled proportionally, resulting in a total domain size of $1.4 \cdot 10^{11}$ cells for the biggest setup. Similar to the results on SuperMUC-NG, near-perfect scalability is achieved. On 1024 NVIDIA GPGPUs, a total performance of $3.9 \cdot 10^3$ GLUPS is demonstrated. The results of the weak scaling analysis are illustrated in Figure 9.10a.



(a) Weak scaling (b) Strong scaling

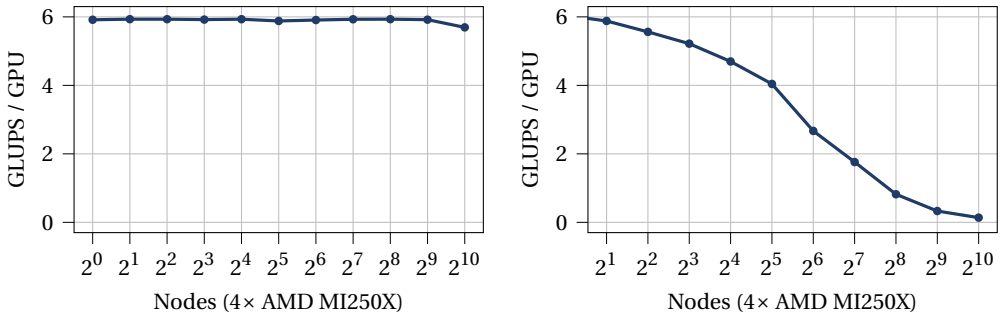
Figure 9.10: Weak (a) and strong (b) scaling behaviour of the lid-driven cavity problem on JUWELS Booster using a uniform mesh resolution.

To analyse the strong scaling behaviour of the simulation, a total domain size of $536.9 \cdot 10^6$ cells is used. This problem size is scaled across four to 1024 GPGPUs. As the number of available hardware resources increases, the performance per GPGPU decreases. Strong scalability eventually falls below 50% at 256 GPGPUs as shown in Figure 9.10b. However, using 1024 GPGPUs, nearly 700 timesteps per second can be processed.

9.5.3 LUMI-G

On the LUMI-G supercomputer, qualitatively similar behaviour is observed as on JUWELS Booster. For the weak scaling benchmark, the same domain size per GPGPU is used, with $134 \cdot 10^6$ cells allocated per GPGPU. In contrast to JUWELS Booster, LUMI-G has a larger number of GPGPUs, allowing the problem to scale from four to 4096 GPGPUs. At this scale, it is possible to run a simulation with a total domain size of $5.5 \cdot 10^{11}$ cells and achieve a total performance of $23.5 \cdot 10^3$ GLUPS, see Figure 9.11a.

For context, some of the largest LBM simulations to date were performed by Liu *et al.* [213]. Their work on the Sunway TaihuLight supercomputer [214] used a uniform grid of $5.6 \cdot 10^{12}$ lattice cells, achieving a total performance of $11.2 \cdot 10^3$ GLUPS. Although the maximum domain size in the present study is approximately ten times smaller, the simulations are nearly twice as fast, making them, to the author's best knowledge, the fastest LBM runs performed to date in terms of total GLUPS.



(a) Weak scaling

(b) Strong scaling

Figure 9.11: Weak (a) and strong (b) scaling behaviour of the lid-driven cavity problem on LUMI-G using a uniform mesh resolution.

A total domain size of $1.07 \cdot 10^6$ cells is allocated to analyse the strong scaling behaviour. This domain is scaled from four to 4096 GPGPUs. As the number of GPGPUs increases, the performance per unit decreases. When more than 256 GPGPUs are used for this problem, the strong scalability falls below 50%, which mirrors the behaviour observed on the JUWELS Booster system. The results are illustrated in Figure 9.11b.

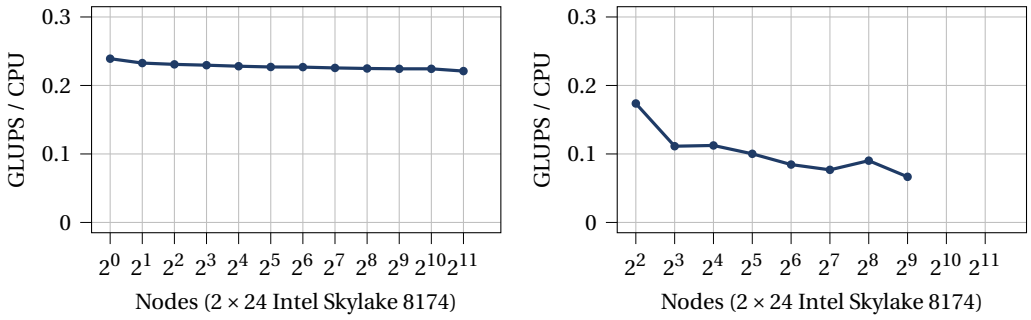
9.6 Scaling Behaviour on Non-uniform Mesh Configurations

Building on the scaling behaviour on a uniform domain, the scaling behaviour of the lid-driven cavity problem is analysed using four mesh levels in the domain. Hence, the interpolations between gird transitions need to be taken into account, and the time-stepping algorithm becomes recursive to realise the temporal refinement as well. This section presents the behaviour of the SRT collision model with a D3Q19 stencil using an Esoteric Twist streaming pattern. Additional benchmarks are gathered Appendix B for more advanced collision models and extended lattice stencils.

9.6.1 SuperMUC-NG

The first setup analysis is the weak scalability on SuperMUC-NG. In this setup, each CPU core manages $3.5 \cdot 10^6$ lattice cells. The simulation is scaled from 48 cores to 98304 cores, resulting in a total domain size of $3.44 \cdot 10^{11}$ lattice cells for the largest setup. As shown in Figure 9.12a, near-perfect weak scaling is achieved. At 98304 cores, the simulation reaches a total performance of $0.91 \cdot 10^3$ GLUPS. Compared to the uniform setup, the non-uniform setup exhibits approximately 16% lower absolute performance due to increased data exchange between the blocks.

This result shows a clear improvement over the previous findings of Schornbaum [29], who reported that the refinement algorithm causes a performance loss of a factor between 2 and 2.5 compared to uniform mesh resolution. However, the results presented here demonstrate a significantly smaller overhead.



(a) Weak scaling

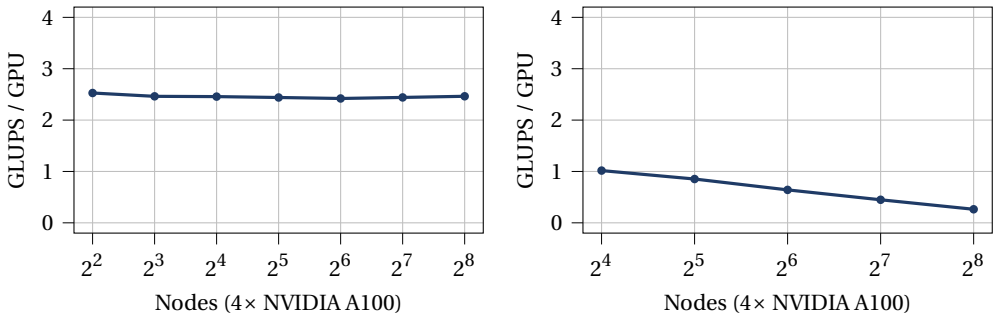
(b) Strong scaling

Figure 9.12: Scaling behaviour of the lid-driven cavity problem on SuperMUC-NG using simulation setup with four different mesh levels. The weak scalability is pictured in (a), while (b) shows the strong scaling capabilities.

For the strong scaling analysis in Figure 9.12b), a total domain size of $3.6 \cdot 10^8$ lattice cells is distributed across four compute nodes. This domain is scaled up to 512 nodes, resulting in a total of 24.576 CPU cores. Despite a performance drop when scaling from four to eight nodes, the simulation demonstrates relatively good strong scalability overall. At 24.576 cores, executing approximately 192 coarse timesteps per second is possible. This is equivalent to 1536 timesteps on the finest mesh level. Unfortunately, runs on higher core counts have shown system errors, and it was possible to repeat the measurements due to time restrictions.

9.6.2 JUWELS Booster

For the weak scaling analysis on JUWELS Booster, each GPGPU holds $112.2 \cdot 10^6$ lattice cells for the largest setup. The problem is scaled from 16 GPGPUs up to 1024 GPGPUs, leading to a total simulation domain size of $1.1 \cdot 10^{11}$ lattice cells. At this scale, the simulation achieves a performance of $2.5 \cdot 10^3$ GLUPS. Similar to the results observed on SuperMUC-NG, near-perfect scalability is realised in this configuration. However, it is notable that the refinement algorithm employed on JUWELS Booster achieves only approximately 64% of the performance compared to the uniform grid setup. This reduction in performance comes from the small compute kernels responsible for the mesh transition. Small compute kernels are inherently bad on GPGPUs and lead to the situation that the hardware can not be saturated. The results are illustrated in Figure 9.13a.



(a) Weak scaling

(b) Strong scaling

Figure 9.13: Weak (a) and strong (b) scaling behaviour of the lid-driven cavity problem on JUWELS Booster using simulation setup with four different mesh levels.

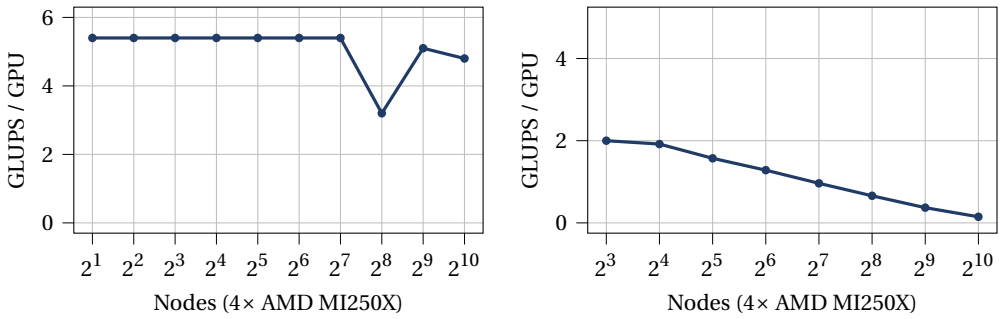
A total domain of $3.6 \cdot 10^9$ lattice cells is used to analyse the strong scaling behaviour. As the number of GPGPUs increases, performance per unit decreases. From 512 GPGPUs, strong scalability drops below 50 %. In this configuration, executing approximately 75 timesteps per second on the coarsest mesh is possible, which translates to about 600 timesteps per second on the finest mesh level. The results are illustrated in Figure 9.13b

Compared to the weak scaling benchmark, this strong scaling benchmark achieves only around 40% of the performance at 64 GPGPUs. This performance discrepancy arises from the benchmark's design and is consistent with the behaviour observed in the strong scaling analysis on SuperMUC-NG. Schornbaum's scaling analysis also reports similar findings [29]. The issue stems from the fact that each process handles between one and four blocks, leading to an inherently unbalanced block structure that cannot be efficiently balanced across all mesh levels. In fact, this setup is only balanced on the finest mesh level, while all other mesh levels remain highly unbalanced.

9.6.3 LUMI-G

Finally, the scaling analysis on non-uniform meshes is extended to LUMI-G. For the weak scaling analysis in Figure 9.14a, $219 \cdot 10^6$ lattice cells are allocated per GPGPU. The benchmark is conducted starting from 8 GPGPUs. Apart from a drop in performance at 1024 GPGPUs, consistent performance is maintained across the range of configurations tested. We could not find a possible explanation for the performance drop at 1024 GPGPUs. However, due to the large scale of this experiment, we also did not repeat the experiment. Overall, a weak scalability of 90% is achieved. At the maximum scale of 4096 GPGPUs, the total simulation domain consists of $1.8 \cdot 10^{12}$ fluid cells, delivering a performance of $19.8 \cdot 10^3$ GLUPS, which is illustrated in Figure 9.14a.

This achievement makes the simulation performed as part of this thesis the largest LBM simulation on non-uniform grids in terms of lattice cells and the fastest in terms of absolute performance that has ever been reported in the literature [23, 24]. Compared to the results of Schornbaum, this work demonstrates a benchmark with more than twice the domain size and over 20 times the performance [29].



(a) Weak scaling

(b) Strong scaling

Figure 9.14: Scaling behaviour of the lid-driven cavity problem on LUMI-G using simulation setup with four different mesh levels. The weak scalability is pictured in (a), while (b) shows the strong scaling capabilities.

For the strong scaling analysis on LUMI-G, a total domain size of $7.2 \cdot 10^9$ lattice cells is used. The benchmark is conducted across a range from 32 GPGPUs to 4096 GPGPUs. As shown in Figure 9.14b, the performance per GPGPU decreases progressively as more hardware is employed. Notably, at 512 GPGPUs, strong scalability drops slightly below 50%. Despite this decline, the benchmark achieves a maximum of 211 timesteps per second on the coarsest mesh, which corresponds to 1688 timesteps per second on the finest mesh level.

9.7 Impact of Numerical Precision

An approach to enhance the runtime performance of a simulation is to reduce the precision of the data arrays and the computations. This reduces the memory footprint and can improve performance for memory-bound compute kernels, such as the LBM. However, it may also result in unacceptable inaccuracies in the simulation results. Therefore, a thorough accuracy analysis is essential before assessing performance. The literature discusses two main approaches for utilising data types of lower numerical precision in the context of the LBM [151]. The first approach is to apply reduced precision generally, meaning that all calculations are performed with reduced precision, and all arrays are stored using a data type with the same numerical precision [60, 215, 216]. The second approach involves storing the PDF array with reduced precision and promoting the array to a higher precision when loading from memory. Then, the collision step is executed, and afterwards, the results are downcasted before storing the PDF array again.

Under this background, we can identify four possibilities which we analyse in this thesis:

DP The PDF array is stored using a 64-bit wide format (double precision), and the collision is executed in the same precision.

DPSP The PDF array is stored using a 32-bit wide format (single precision), and the collision is executed using double precision. Hence, data casts are inserted when loading and storing the PDF array.

SP The PDF array is stored using a 32-bit wide format, and the collision is executed with the same precision.

SPHP The PDF array is stored using a 16-bit wide format (half precision), and the collision is executed using single precision. Hence, data casts are inserted when loading and storing the PDF array.

While many possibilities exist to represent floating point number formats, we will only focus on IEEE-754 formats here [217]. Custom formats for the LBM with 16-bit wide floating point have been recently introduced by Lehmann *et al.* [151]. Hence, the interested reader is referred to their work for more information.

In order to assess the impact of the used floating point precision on the numerical accuracy, we replicate a test case recently published in [151] involving the Taylor-Green vortex flow. In this work, however, we use a three-dimensional version of the Taylor-Green vortex problem. This is achieved by extending the two-dimensional problem of Lehmann *et al.* [151] in the z-direction with a zero velocity in the third dimension. The Taylor-Green vortex setup consists of a periodic box of vortices and is initialised with a velocity magnitude of u_0. The transient decay of the vortices is simulated and compared to the known analytic solution. The analytic solution, which also specifies the initial flow field, reads

$$\begin{aligned} u_x(\boldsymbol{x}, t=0) &= u_0 \cos(\kappa\, x) \sin\left(\kappa\, y\right) \exp\left(-2\nu\kappa^2 t\right), \\ u_y(\boldsymbol{x}, t=0) &= -u_0 \sin(\kappa\, x) \cos\left(\kappa\, y\right) \exp\left(-2\nu\kappa^2 t\right), \\ u_z(\boldsymbol{x}, t=0) &= 0, \\ \rho\,(\boldsymbol{x}, t=0) &= 1 - \frac{3\,u_0^2}{4}\left(\cos(2\kappa\, x) + \cos\left(2\kappa\, y\right)\right) \exp\left(-4\nu\kappa^2 t\right). \end{aligned} \tag{9.3}$$

The system is initialised at $t = 0$ with $u_0 = 0.25$. We set the kinematic shear viscosity to $\nu = \frac{1}{6}$, which results in the viscosity-governing relaxation rate $\omega = 1$. Furthermore, $\kappa = \frac{2\pi}{L}$ and $L = 128$ is the side length of the squared domain. The initialisation of the benchmark is pictured in Figure 9.15.

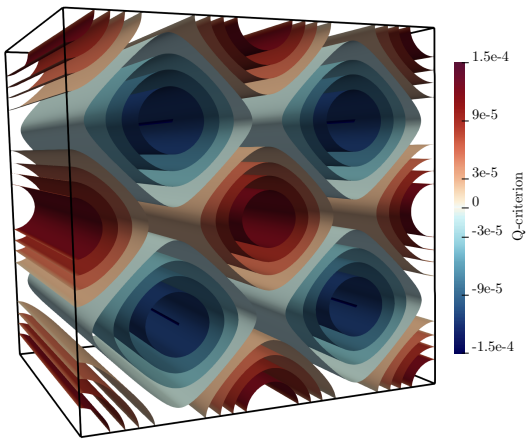


Figure 9.15: Initialisation of the Taylor-Green vortex benchmark with $u_0 = 0.25$, $\nu = \frac{1}{6}$, $\omega = 1$, $\kappa = \frac{2\pi}{L}$ and $L = 128$. The figure shows the Q-criterion of the velocity at the initial timestep.

When using reduced precision, it is essential to reduce the round-off errors which occur in the calculation of the collision step of the LBM. One of the most important optimisations in this regard has been introduced by Skordos [141]. His work aimed to identify a background density ρ_0 that can be subtracted from the PDFs. In the scope of this thesis, we have introduced this optimisation in LBMPY for all collision models [1]. In the remainder of this section, only optimised compute kernels are shown. This means compute kernels, where the PDFs fluctuate around zero.

In Figure 9.16, the analytical solution for the kinetic energy (compare Equation (9.3)) is compared with the simulated results. Due to viscous friction, the vortex velocity and, consequently, the kinetic energy is expected to decay exponentially. The simulation accurately reflects this behaviour until, at a certain point, the simulated energy levels off onto a plateau. This phenomenon arises because lower velocities can no longer be accurately represented, owing to truncation errors introduced by the floating-point number format. For the IEEE-754 double precision format [217], the truncation error is $\epsilon = 2.2 \cdot 10^{-16}$. Given the squared calculation of the kinetic energy, the plateau is expected to occur at approximately ϵ^2. Here, the kinetic energy is calculated as

$$E(t) = \int_0^L \int_0^L \int_0^L \frac{\rho}{2} \left(u_x^2 + u_y^2 + u_z^2 \right) dx\, dy\, dz = u_0^2 \pi^2 \exp\left(-4\nu\kappa^2 t\right), \tag{9.4}$$

and we denote the initial kinetic energy as $E_0 = E(0)$. Several important results can be taken from Figure 9.16. First, using double precision for the collision step and the storage of the PDF array achieves the lowest plateau and, hence, the highest accuracy. Second, using a mixed format (DPSP) achieves almost similar accuracy, while a pure single precision compute kernel (SP) leads to much lower accuracy. Using single-precision computations and storing the PDF array in half-precision (SPHP) does not lead to the same improvement as the single-, double-precision counterpart (DPSP). Hence, it indicates that half-precision truncates the PDF array too severely. Lastly, an important insight is that both the SRT and cumulant collision model achieve similar accuracy in all benchmarks.

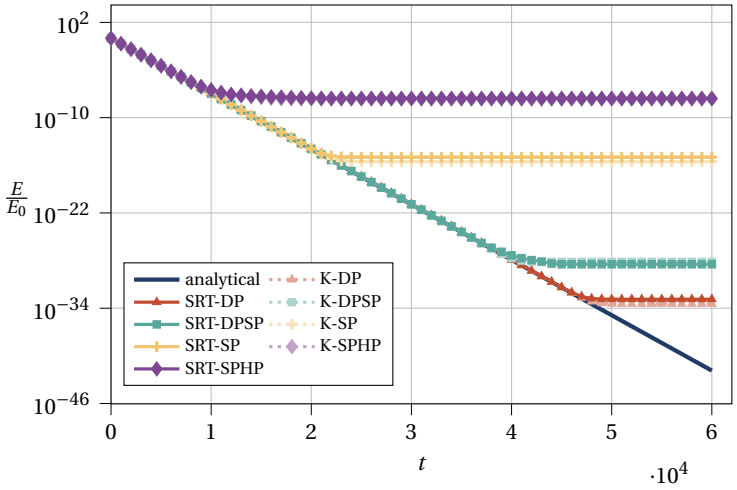


Figure 9.16: Relative energy $E(t)/E_0$ for various numerical precisions (DP, DPSP, SP, and SPHP) and collision models.

After analysing the accuracy of the compute kernels, we analyse their performance on an NVIDIA H100 GPGPU. This is shown in Figure 9.17. All compute kernels use a D3Q19 lattice stencil and the pull streaming pattern. Due to the memory-bound nature of the LBM, it can be seen that both SP and DPSP can achieve double the performance of a pure double precision kernel (DP). In this version, only half of the memory needs to be transferred from the main memory, which explains the speedup. Furthermore, the mixed half-precision kernel can achieve another speedup. However, the speedup is less significant than the jump from double to single precision.

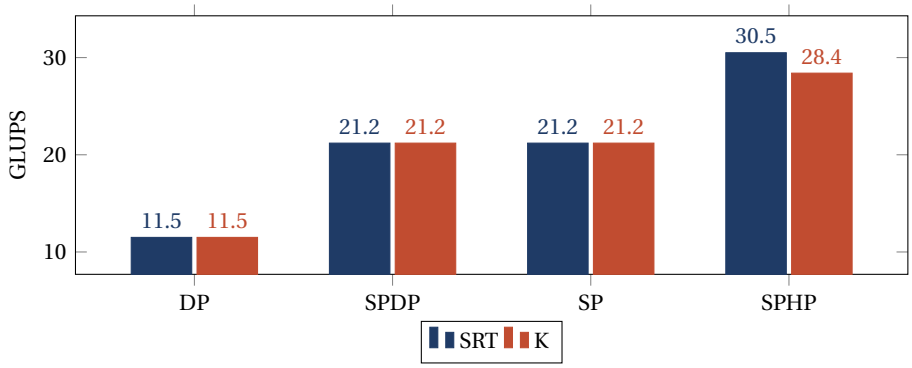


Figure 9.17: Comparision of the performance of generated compute kernels for the SRT and the cumulant collision model using a D3Q19 lattice stencil and the pull streaming pattern on an NVIDIA H100 GPGPU. The compute kernels use different numerical precision to compute the collision step and store the PDF array.

The performance improvements shown in Figure 9.17 make using a lower data format an attractive option. However, it must be emphasised that numerous remaining questions still need to be answered in this direction. First of all, we have only analysed a simple benchmark case here. Hence, the influence on complex single-phase or multiphase flows still needs to be shown. One important aspect is that the optimisation by Skordos [141] is designed explicitly for hydrodynamic LBMs. This means LBMs that recovers the Navier-Stokes equations (NSEs). For other equations (e.g., the Allen-Cahn equation (ACE)) it might be less effective because the background density undergoes higher fluctuations. Furthermore, we have not analysed the accuracy of highly turbulent flows on refined meshes. However, the drag-crisis, as shown in Chapter 6, was first recovered by Geier *et al.* using only single precision [60]. This indicates that pure single-precision calculations might be enough even for these cases.

CHAPTER 10

CONCLUSIONS & PERSPECTIVES

10.1 Results Summary

The exascale computing era is driven by specialised hardware, continuing the trend of advancing Moore's Law. While heterogeneous supercomputers have surpassed the exaFLOP threshold, efficiently utilising these resources presents a significant challenge for application developers. Code generation techniques offer promising solutions to address that issue. In this research, we have explored these techniques to enable large-scale simulations using the lattice Boltzmann method (LBM). The toolchain we have developed here is structured at distinct levels and serves a clear, distinct purpose.

While the hardware landscape becomes increasingly complex, numerical solvers are naturally complicated. The LBM is no exception. Over the past decades, increasingly sophisticated collision models have been proposed to extend the LBM, e.g., for highly turbulent or multiphase flow conditions. In this thesis, we have significantly enhanced LBMPY to describe and generate these advanced models. This involved utterly restructuring the software to modularise the description and derivation of LBMs. The resulting state-of-the-art compute kernels demonstrate an especially low number of floating point operations per second (FLOPS). Additionally, we have optimised and extended the collision models to support multiphase and thermocapillary flows. Furthermore, we have added recently formulated boundary conditions, which show greater accuracy around complex geometries. These developments make LBMPY an increasingly useful tool for computer science, physics and research.

The resulting symbolic description of the LBM can be transformed into highly optimised, hardware-specific compute kernels using the PYSTENCILS package. For this code generation package, we have made essential advancements. We have extended it to support a broader range of hardware. On the central processing unit (CPU) side, this includes expanding support for single instruction, multiple data (SIMD) instruction sets, such as those used by ARM CPUs. On the accelerator side, we have added support for AMD general purpose graphics processing units (GPGPUs), which are extensively utilised in the world's largest supercomputers at the time of writing.

Finally, the extensions to the code generator toolchain have resulted in highly optimised compute kernels that have been integrated into the WALBERLA framework. The closer integration of the code generator with WALBERLA that we have developed has enabled complex enhancements that significantly boost WALBERLA's capabilities. This includes a complete overhaul of the mesh refinement algorithm, now based on the code generator. By leveraging PYSTENCILS and LBMPY to produce optimised compute kernels at the grid interface, a new algorithm has been developed that is independent of the LBM streaming pattern and requires fewer ghost layers around WALBERLA's block structures. In this research, we have implemented the new refinement algorithm in WALBERLA. This implementation is compatible with CPU and GPGPU architectures.

The enhanced capabilities of the code generation toolchain, combined with WALBERLA, have been demonstrated in this thesis through numerous complex applications. We have used the new refinement algorithm to replicate the drag crisis of a spherical object in a simulation setup using GPGPUs. The drag crisis occurs under highly turbulent flow conditions, which can only be simulated through the extension of LBMPY with state-of-the-art collision models and boundary treatments. This achievement underscores the synergy between the code generation framework and the domain decomposition capabilities of WALBERLA.

Additionally, we have analysed multiphase slug flow problems in a complex annulus pipe configuration. These simulations demand a high level of detail, necessitating a supercomputer. The results are consistent with the existing literature for base cases and extend the literature to more complex setups. Building on the newly integrated multiphase capabilities, we have examined thermocapillary flows in a three-dimensional configuration. The simulation results indicate that two-dimensional simulations are inadequate for engineering design decisions, highlighting the necessity of three-dimensional simulations, which again require highly optimised compute kernels.

We have conducted a comprehensive benchmark analysis to assess the computational performance and efficiency of the developments achieved in this thesis. We could demonstrate consistent performance across various CPU and GPGPU platforms, starting from a single chip. Our analysis includes the latest hardware used in the most powerful supercomputers at the time of writing. Additionally we have compared the measured performance results to a suitable performance model in order to be able to classify them. For uniform and non-uniform mesh configurations, we investigated scaling from this baseline. These results provide valuable insights into various aspects, allowing researchers to estimate computational overhead arising from factors such as the streaming pattern, collision model, hardware, or mesh configuration. To the author's knowledge, the scaling runs represent some of the largest simulations performed with the LBM on uniform mesh configurations and the largest on non-uniform configurations to date. This clearly shows that WALBERLA is ready for the exascale era.

We have added all concepts and algorithms which have been presented in this thesis to PYSTENCILS, LBMPY and WALBERLA. The benchmark cases for the uniform and the non-uniform setup and all application codes are available and can be downloaded freely. Hence, all work presented here is available under an open-source licence because all individual frameworks are open-source.

10.2 Future Work

The research we have presented here opens the door for numerous future developments. In this work, we have extended WALBERLA for AMD GPGPUs. However, at this point, Intel also started to release new types of GPGPUs. In fact the second largest supercomputer (according to the Top500 of June 2024[1]), Aurora[2] is powered by Intel GPGPUs. Remarkably, the Aurora system also breaks the exaFLOPS barrier, which shows the importance of supporting Intel GPGPUs in the future.

To support this architecture, PYSTENCILS might need a new backend, which introduces relevant optimisations. However, WALBERLA needs to be extended to integrate the new architecture into the framework. This could be realised by employing recent developments of the C++ standard library or using a library like SYCL[3].

Either way, a stronger integration of accelerator hardware in WALBERLA would be desirable in general. Due to the developments in the code generation pipeline, the integration of accelerators was generally kept thin. This was favourable because it resulted in a relatively simple framework structure. However, it prohibits more complex algorithms like adaptive mesh refinement. In these scenarios, the mesh structure changes during the simulation. Thus, ideally, the octree structure is available for accelerators to some extent. Similarly, this would open the door to removing data exchanges for blocks on the same processor. If the octree structure is known on the accelerator, data of neighbouring blocks could be fetched directly instead of using additional data passes within the pack info.

In this work, we have shown that we can successfully recover the drag crisis of a spherical object. Here, we used a rather simple scheme to transport data between grid levels. An interesting future work would be to use more complex schemes like the compact interpolation of [149].

The multiphase simulations we have conducted in this thesis show interesting insights into the behaviour of Taylor bubbles in complex pipe configurations. However, it was not possible to study the interaction of Taylor bubbles with each other due to stability issues in these simulations. This opens the door for future improvements. One idea might be to use a pressure-based cumulant formulation for the hydrodynamic solver, as shown by Sitompul and Aoki [218]. However, their work also used filtering techniques on the density and velocity field. It would be interesting to analyse how such techniques influence the accuracy of the Taylor bubble simulations. Another idea might be using adaptive mesh refinement, which goes along with the earlier points. A starting point for adaptive mesh refinement with the conservative Allen-Cahn model (CACM) is given by Fakhari *et al.* [50].

The thermocapillary studies we have shown in this thesis are proof that our developments enable us to recover the correct behaviour of thermocapillary systems. However, so far, we have only looked at the general behaviour and did not compare our simulations with three-dimensional experimental data. A recent article that delivers experimental data for interesting thermocapillary systems was published by Won *et al.* [204]. Following their work, the thermocapillary algorithm of WALBERLA could be validated with experimental data.

[1] https://top500.org/lists/top500/2024/06/
[2] https://www.anl.gov/aurora
[3] https://www.khronos.org/sycl/

Another interesting aspect would be the analysis of the energy consumption of the generated compute kernels. For handwritten kernels, Wittmann *et al.* [212] argues that a low computational intensity can lead to energy savings because it can allow lowering the clock speed of a CPU without sacrificing much performance. While efforts in this direction have been started in this thesis [7], it would be interesting to perform a systematic analysis of the various collision models that LBMPY supports and to show the effect of the sophisticated mathematical optimisations that are applied. Furthermore, this work could be performed on different state-of-the-art architectures to showcase the impact in various scenarios.

APPENDIX A

ANALYTIC SOLUTION OF PLANAR HEATED CHANNEL

The analytical solution for thermocapillary-driven convection of superimposed fluids was derived in the work of Pendse *et al.* [201]. In this chapter, we present the result for the temperature and the velocity field along with the necessary assumptions for the solution to hold. First, it is important to define the flow regime which is analysed here. Typical applications that lay the foundation of the analytical analysis of Pendse *et al.* are in the size of $L = 100 \cdot 10^{-6}\,\mathrm{m}$ and $H = 50 \cdot 10^{-6}\,\mathrm{m}$ for the length and the height of the channel respectively. Furthermore, they considered temperature gradients in the range of $|\nabla T| = 1 \cdot 10^{-2}\,\mathrm{K\,m^{-1}}$, surface tension gradients in the range of $\sigma_\mathrm{T} = -1 \cdot 10^{-4}\,\mathrm{N\,m^{-1}\,K}$, a surface tension in the range of $\sigma_\mathrm{ref} = 0.3\,\mathrm{N\,m^{-1}}$, a kinematic viscosity in the range of $\nu = 1 \cdot 10^{-5}\,\mathrm{m^2\,s^{-1}}$, and a thermal diffusivity in the range of $\kappa = 1 \cdot 10^{-5}\,\mathrm{m^2\,s^{-1}}$. These length scales and fluid properties lead to typical dimensionless numbers of $\mathrm{Re} \ll 1$, $\mathrm{Ma} \ll 1$, and $\mathrm{Ca} \ll 1$, respectively.

The analytical solution is then derived from the governing equations. These are the conservation of mass and momentum and the balance of thermal energy at the steady state. Since the considered fluids are incompressible, immiscible, and Newtonian, and the dimensionless numbers are small, it is possible to ignore the convective transport of momentum and energy. Furthermore, through $\mathrm{Ca} \ll 1$ it can be concluded that the interface remains flat, which further simplifies the problem. The resulting simplified governing equations are

$$\nabla \boldsymbol{u} = 0, \tag{A.1}$$

$$-\nabla p + \mu \nabla^2 \boldsymbol{u} = 0, \tag{A.2}$$

$$\nabla^2 T = 0. \tag{A.3}$$

Notably, the LBM allows for small fluctuations of the density and, thus, the incompressibility condition can not be fulfilled perfectly. This is generally problematic for the CACM and improvements are made by the pressure formulation of the hydrodynamic solver as shown in Section 3.3.2. Furthermore, the equations are coupled together with jump conditions which are not exactly represented by the smooth interface that appears in phase-field models like the CACM. Building on these assumptions, the boundary conditions (see

Equations (8.1) and (8.2)), and the assumptions that the temperature and the heat flux must be continuous at the interface, it is possible to derive an analytic solution for the temperature field. It reads,

$$T(x,y) = T(x,y)' + \widetilde{T(y)}, \tag{A.4}$$

where

$$\widetilde{T(y)} = \begin{cases} \dfrac{\kappa_{\mathrm{H}}(T_c - T_h)y + \kappa_{\mathrm{L}} T_c b + \kappa_{\mathrm{H}} T_h a}{a\kappa_{\mathrm{H}} + \kappa_{\mathrm{L}} b}, & y > 0, \\ \dfrac{\kappa_{\mathrm{L}}(T_c - T_h)y + \kappa_{\mathrm{L}} T_c b + \kappa_{\mathrm{H}} T_h a}{a\kappa_{\mathrm{H}} + \kappa_{\mathrm{L}} b}, & y \leq 0, \end{cases} \tag{A.5}$$

is the linear temperature field and

$$T(x,y)' = \begin{cases} T_0 f(a_r, b_r, \kappa^*) \sinh\big(a_r - \omega y\big)\cos(\omega x), & y > 0, \\ T_0 f(a_r, b_r, \kappa^*) \left[\sinh(a_r)\cosh\big(\omega y\big) - \kappa^* \sinh\big(\omega y\big)\cosh(a_r)\right]\cos(\omega x), & y \leq 0, \end{cases} \tag{A.6}$$

is the perturbation temperature field and $a = b = H/2$[1]. Furthermore, the unknown parameters are defined as

$$a_r = a\omega, \qquad b_r = b\omega, \tag{A.7}$$

$$f(a_r, b_r, \kappa^*) = \frac{1}{\kappa^* \sinh(b_r)\cosh(a_r) + \sinh(a_r)\cosh(b_r)}. \tag{A.8}$$

Similarly, Pendse *et al.* [201] found a solution for the velocity field which reads,

$$u(x,y) = U_m\{[C_1 + \omega(C_2 + C_3 y)]\cosh\big(\omega y\big) + (C_3 + \omega C_1 y)\sinh\big(\omega y\big)\}\sin(\omega x), \tag{A.9}$$

$$v(x,y) = -\omega U_m[C_1 y\cosh\big(\omega y\big) + (C_2 + C_3 y)\sinh\big(\omega y\big)] \times \cos(\omega x), \tag{A.10}$$

where the x- and y-components of velocity are given by u and v, respectively. The additional parameters in the previous equations are expressed as,

$$C_1 = \begin{cases} \dfrac{\sinh^2(a_r)}{\sinh^2(a_r) - {a_r}^2} & y > 0, \\ \dfrac{\sinh^2(b_r)}{\sinh^2(b_r) - {b_r}^2} & y \leq 0 \end{cases} \qquad C_2 = \begin{cases} \dfrac{-a a_r}{\sinh^2(a_r) - {a_r}^2} & y > 0, \\ {C_2}^b = \dfrac{-b b_r}{\sinh^2(b_r) - {b_r}^2} & y \leq 0 \end{cases} \tag{A.11}$$

$$C_3 = \begin{cases} \dfrac{2a_r - \sinh(2a_r)}{2[\sinh^2(a_r) - {a_r}^2]} & y > 0, \\ \dfrac{-2b_r + \sinh(2b_r)}{2[\sinh^2(b_r) - {b_r}^2]} & y \leq 0 \end{cases} \tag{A.12}$$

$$U_m = -\left(\frac{T_0 \sigma_T}{\mu_h}\sinh a_r f(a_r, b_r, \kappa^*) h(a_r, b_r, \mu^*)\right), \tag{A.13}$$

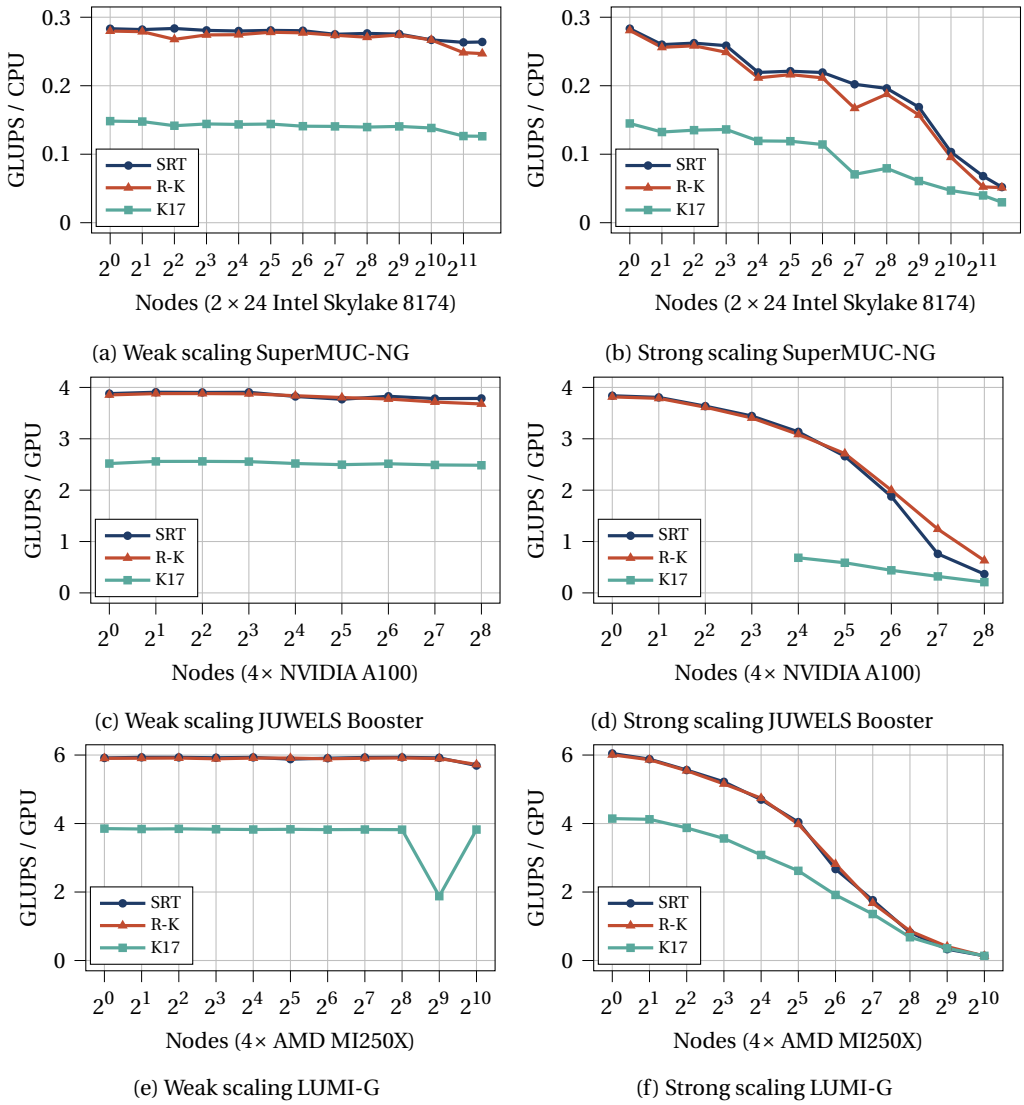
$$h(a_r, b_r, \mu^*) = \frac{\left(\sinh^2(a_r) - {a_r}^2\right)\left(\sinh^2(b_r) - {b_r}^2\right)}{\mu^*\left(\sinh^2(b_r) - {b_r}^2\right)(\sinh(2a_r) - 2a_r) + \left(\sinh^2(a_r) - {a_r}^2\right)(\sinh(2b_r) - 2b_r)}. \tag{A.14}$$

[1]The values for a and b need to be adapted respectively if the interface of the fluids is not exactly dividing the channel in half

APPENDIX B

EXTENDED SCALING ANALYSIS

Extended scaling results are presented in this section. The benchmarks shown here share the same configuration as reported in Section 9.5 and Section 9.6. However, additional collision models are added to show their scaling behaviour in comparison to the single-relaxation-time (SRT) collision model. The first collision setup is the regularised cumulant collision operator (R-K) using a D3Q19 lattice stencil, and the second model is the K17 collision operator, which uses a D3Q27 lattice stencil. The two collision models are chosen because these are the most complex models for the respective lattice stencil. This means they show the most FLOPSs.



(a) Weak scaling SuperMUC-NG

(b) Strong scaling SuperMUC-NG

(c) Weak scaling JUWELS Booster

(d) Strong scaling JUWELS Booster

(e) Weak scaling LUMI-G

(f) Strong scaling LUMI-G

Figure B.1: Weak and strong scaling benchmarks of various collision models on SuperMUC-NG (a) and (b), JUWELS Booster (c) and (d), and LUMI-G (e) and (f). All benchmarks use a uniform mesh resolution.

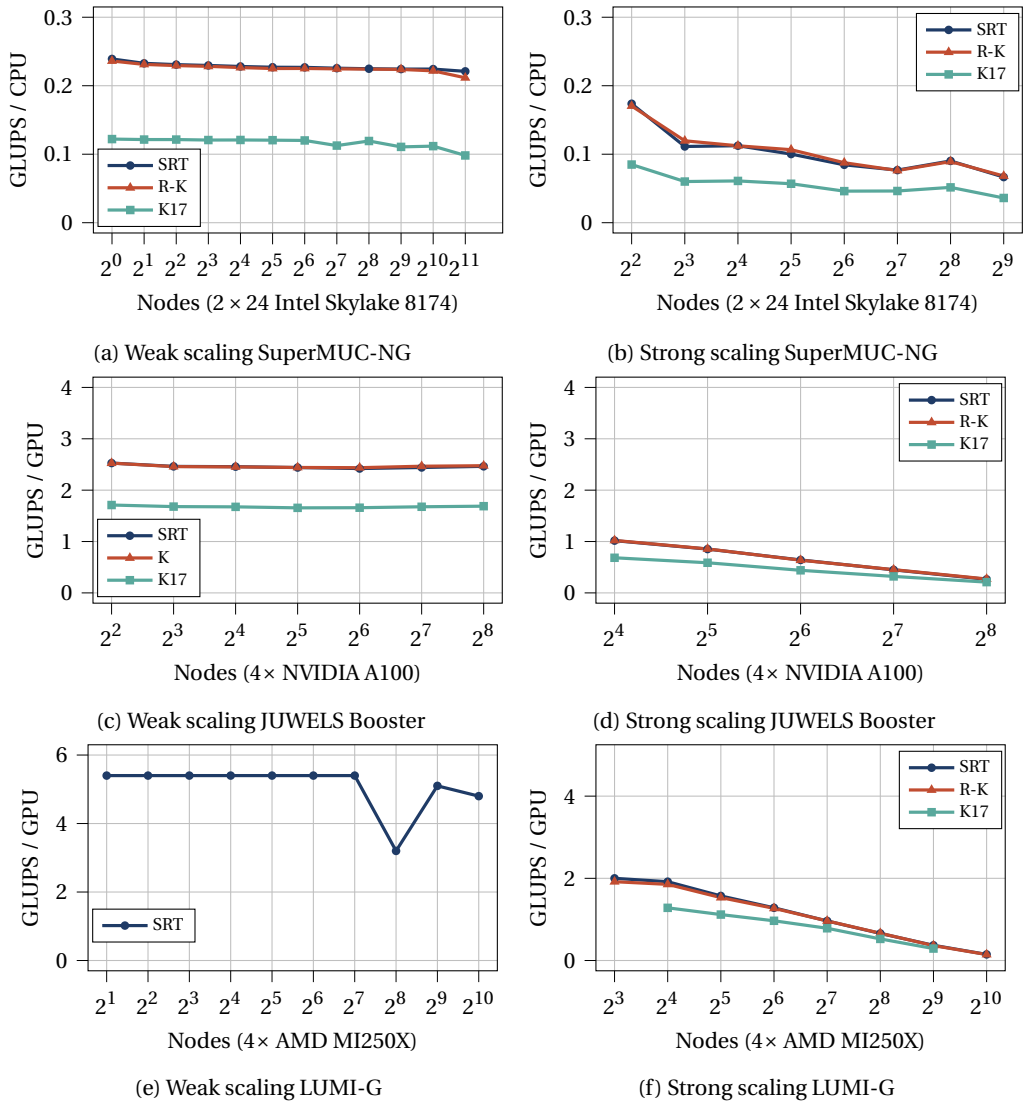


Figure B.2: Weak and strong scaling benchmarks of various collision models on SuperMUC-NG (a) and (b), JUWELS Booster (c) and (d), and LUMI-G (e) and (f). All benchmarks use a non-uniform mesh resolution.

BIBLIOGRAPHY

(Co-)Authored Publications

[1] F. Hennig, M. Holzer, and U. Rüde, "Advanced Automatic Code Generation for Multiple Relaxation-Time Lattice Boltzmann Methods", *SIAM Journal on Scientific Computing*, vol. 45, no. 4, pp. C233–C254, Aug. 2023.

[2] T. Mitchell, M. Holzer, C. Schwarzmeier, M. Bauer, U. Rüde, and C. Leonardi, "Stability assessment of the phase-field lattice Boltzmann model and its application to Taylor bubbles in annular piping geometries", *Physics of Fluids*, vol. 33, no. 8, Aug. 2021.

[3] C. Schwarzmeier, M. Holzer, T. Mitchell, M. Lehmann, F. Häusl, and U. Rüde, "Comparison of free-surface and conservative Allen–Cahn phase-field lattice Boltzmann method", *Journal of Computational Physics*, vol. 473, p. 111 753, Jan. 2023.

[4] M. Holzer, M. Bauer, H. Köstler, and U. Rüde, "Highly efficient lattice Boltzmann multiphase simulations of immiscible fluids at high-density ratios on CPUs and GPUs through code generation", *The International Journal of High Performance Computing Applications*, vol. 35, no. 4, pp. 413–427, May 2021.

[5] M. Holzer, T. R. Mitchell, C. R. Leonardi, and U. Rüde, "Development of a central-moment phase-field lattice Boltzmann model for thermocapillary flows: droplet capture and computational performance", *Journal of Computational Physics*, p. 113 337, Aug. 2024.

[6] D. Ernst, M. Holzer, G. Hager, M. Knorr, and G. Wellein, "Analytical performance estimation during code generation on modern GPUs", *Journal of Parallel and Distributed Computing*, vol. 173, pp. 152–167, Mar. 2023.

[7] O. Vysocky, M. Holzer, G. Staffelbach, R. Vavrik, and L. Riha, "Energy-Efficient Implementation of the Lattice Boltzmann Method", *Energies*, vol. 17, no. 2, p. 502, Jan. 2024.

[8] M. Holzer, G. Staffelbach, I. Rocchi, J. Badwaik, A. Herten, R. Vavrik, O. Vysocky, L. Riha, R. Cuidard, and U. Ruede, "Scalable Flow Simulations with the Lattice Boltzmann Method", in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23, ACM, May 2023.

Supervised Theses

[9] M. Warlich, "Implementation of Communication Schemes for the LB Method using Different Data Structures through Meta-ProgrammingTechniques", BA thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2021.

[10] D. Bauer, “Flexible Lattice Boltzmann Simulation Pipeline for Airborne SARS-CoV-2 Transmission with waLBerla and Blender”, M.S. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2020.

[11] F. Hennig, “Streaming-Pattern-Independent Realization of Lattice Boltzmann Grid Refinement as an MPI-based Communication Scheme via Code Generation”, BA thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2021.

Other References

[12] *The Nobel Prize in Physics 2023*, Accessed: 2024-06-25, 2023.

[13] B. H. McCarthy and S. Ponedal, *IBM Unveils World’s First 2 Nanometer Chip Technology, Opening a New Frontier for Semiconductors*, Accessed: 2024-06-25, 2021.

[14] D. Ielmini and H.-S. P. Wong, “In-memory computing with resistive switching devices”, *Nature Electronics*, vol. 1, no. 6, pp. 333–343, Jun. 2018.

[15] S. Alam, J. Hutchins, N. Shukla, K. Asifuzzaman, and A. Aziz, “CMOS-Based Single-Cycle in-Memory XOR/XNOR”, *IEEE Access*, vol. 12, pp. 49 528–49 534, 2024.

[16] B. Hendrickson, A. Aceves, E. Alhajjar, M. Bañuelos, D. Brown, K. Devine, Q. Du, O. Ghattas, R. Giles, M. Hall, T. Islam, K. Jordan, L. Lin, A. Pothen, P. Raghavan, R. Schreiber, C. Thalhauser, and A. Wilson, “The future of computational science”, SIAM Society for Industrial and Applied Mathematics, Tech. Rep., 2024.

[17] A. Herten, “Many Cores, Many Models: GPU Programming Model vs. Vendor Compatibility Overview”, in *Proceedings of the SC ’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W 2023, ACM, Nov. 2023.

[18] J. H. Ferziger, *Computational Methods for Fluid Dynamics*, third, rev. edition, M. Perić, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, 42649 pp.

[19] H. Chen, S. Chen, and W. H. Matthaeus, “Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method”, *Physical Review A*, vol. 45, no. 8, R5339–R5342, Apr. 1992.

[20] C. K. Aidun and J. R. Clausen, “Lattice-Boltzmann Method for Complex Flows”, *Annual Review of Fluid Mechanics*, vol. 42, no. 1, pp. 439–472, Jan. 2010.

[21] P. Lallemand, L.-S. Luo, M. Krafczyk, and W.-A. Yong, “The lattice Boltzmann method for nearly incompressible flows”, *Journal of Computational Physics*, vol. 431, p. 109 713, Apr. 2021.

[22] C. Godenschwager, F. Schornbaum, M. Bauer, H. Köstler, and U. Rüde, “A framework for hybrid parallel flow simulations with a trillion cells in complex geometries”, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC13, ACM, Nov. 2013.

[23] F. Schornbaum and U. Rüde, “Massively Parallel Algorithms for the Lattice Boltzmann Method on NonUniform Grids”, *SIAM Journal on Scientific Computing*, vol. 38, no. 2, pp. C96–C126, Jan. 2016.

[24] F. Schornbaum and U. Rüde, “Extreme-Scale Block-Structured Adaptive Mesh Refinement”, *SIAM Journal on Scientific Computing*, vol. 40, no. 3, pp. C358–C387, Jan. 2018.

[25] M. Bauer, S. Eibl, C. Godenschwager, N. Kohl, M. Kuron, C. Rettinger, F. Schornbaum, C. Schwarzmeier, D. Thönnes, H. Köstler, and U. Rüde, "waLBerla: A block-structured high-performance framework for multiphysics simulations", *Computers & Mathematics with Applications*, vol. 81, pp. 478–501, Jan. 2021.

[26] M. Bauer, H. Köstler, and U. Rüde, "Lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods", *Journal of Computational Science*, vol. 49, p. 101 269, Feb. 2021.

[27] M. Holzer, "Efficient Code Generation of a Lattice Boltzmann Phase-Field Model for Immiscible Fluids with High Density Ratios and High Reynolds Numbers", M.S. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2020.

[28] M. Bauer, J. Hötzer, D. Ernst, J. Hammer, M. Seiz, H. Hierl, J. Hönig, H. Köstler, G. Wellein, B. Nestler, and U. Rüde, "Code generation for massively parallel phase-field simulations", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, ACM, Nov. 2019.

[29] F. Schornbaum, "Block-Structured Adaptive Mesh Refinement for Simulations on Extreme-Scale Supercomputers", Ph.D. dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2018.

[30] T. Gruber, J. Hammer, and J. Laukemann, *RRZE-HPC/MachineState: MachineState-0.4.1*, 2021.

[31] J. C. Maxwell, "The dynamical theory of gases", *Philosophical Transactions of the Royal Society of London*, 1867.

[32] L. Boltzmann and S. G. Brush, "Weitere Studien über das Wärmegleichgewicht unter Gasmolekülen", in *Kinetische Theorie II: Irreversible Prozesse Einführung und Originaltexte*, Wiesbaden: Vieweg+Teubner Verlag, 1970, pp. 115–225.

[33] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen, *The Lattice Boltzmann Method: Principles and Practice*. Springer International Publishing, 2017.

[34] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems", *Physical Review*, vol. 94, no. 3, pp. 511–525, May 1954.

[35] Y. H. Qian, D. D'Humières, and P. Lallemand, "Lattice BGK Models for Navier-Stokes Equation", *Europhysics Letters (EPL)*, vol. 17, no. 6, pp. 479–484, Feb. 1992.

[36] C. Coreixas, B. Chopard, and J. Latt, "Comprehensive comparison of collision models in the lattice Boltzmann framework: Theoretical investigations", *Physical Review E*, vol. 100, no. 3, p. 033 305, Sep. 2019.

[37] M. Wittmann, T. Zeiser, G. Hager, and G. Wellein, "Comparison of different propagation steps for lattice Boltzmann methods", *Computers $ Mathematics with Applications*, vol. 65, no. 6, pp. 924–935, Mar. 2013.

[38] P. Bailey, J. Myre, S. D. Walsh, D. J. Lilja, and M. O. Saar, "Accelerating lattice Boltzmann fluid flow simulations using graphics processors", in *2009 International Conference on Parallel Processing*, IEEE, Sep. 2009.

[39] M. Geier and M. Schönherr, "Esoteric Twist: An efficient in-place streaming algorithmus for the lattice Boltzmann method on massively parallel hardware", *Computation*, vol. 5, no. 4, p. 19, Mar. 2017.

[40] M. Lehmann, "Esoteric pull and Esoteric push: Two simple in-place streaming schemes for the lattice Boltzmann method on GPUs", *Computation*, vol. 10, no. 6, p. 92, Jun. 2022.

[41] A. Kummerländer, M. Dorn, M. Frank, and M. J. Krause, "Implicit propagation of directly addressed grids in lattice Boltzmann methods", *Concurrency and Computation: Practice and Experience*, vol. 35, no. 8, Feb. 2023.

[42] A. Perepelkina, V. Levchenko, and A. Zakirov, "New Compact Streaming in LBM with ConeFold LRnLA Algorithms", in *Supercomputing*. Springer International Publishing, 2020, pp. 50–62.

[43] D. d'Humières, "Generalized Lattice-Boltzmann Equations", in *Rarefied Gas Dynamics: Theory and Simulations*. American Institute of Aeronautics and Astronautics, Jan. 1992, pp. 450–458.

[44] D. d'Humières, "Multiple–relaxation–time lattice Boltzmann models in three dimensions", *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 360, no. 1792, P. V. Coveney, S. Succi, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo, Eds., pp. 437–451, Mar. 2002.

[45] P. J. Dellar, "Bulk and shear viscosities in lattice Boltzmann equations", *Physical Review E*, vol. 64, no. 3, p. 031 203, Aug. 2001.

[46] P. Lallemand and L.-S. Luo, "Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability", *Physical Review E*, vol. 61, no. 6, pp. 6546–6562, Jun. 2000.

[47] L. N. Trefethen, *Numerical linear algebra*, D. Bau, Ed. Philadelphia, Pa.: Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997, 1361 pp., Includes bibliographical references and index. - Title from title screen, viewed 12/30/2010.

[48] P. J. Dellar, "Nonhydrodynamic modes and a *priori* construction of shallow water lattice Boltzmann equations", *Physical Review E*, vol. 65, no. 3, p. 036 309, Feb. 2002.

[49] M. Geier, M. Schönherr, A. Pasquali, and M. Krafczyk, "The cumulant lattice Boltzmann equation in three dimensions: theory and validation", *Computers and Mathematics with Applications*, vol. 70, no. 4, pp. 507–547, Aug. 2015.

[50] A. Fakhari, D. Bolster, and L.-S. Luo, "A weighted multiple-relaxation-time lattice Bltzmann method for multiphase flows and its application to partial coalescence cascades", *Journal of Computational Physics*, vol. 341, pp. 22–43, Jul. 2017.

[51] M. Geier, A. Greiner, and J. G. Korvink, "Cascaded digital lattice Boltzmann automata for high Reynolds number flow", *Physical Review E*, vol. 73, no. 6, p. 066 705, Jun. 2006.

[52] K. N. Premnath and S. Banerjee, "On the Three-Dimensional Central Moment Lattice Boltzmann Method", *Journal of Statistical Physics*, vol. 143, no. 4, pp. 747–794, May 2011.

[53] P. Asinari, "Generalized local equilibrium in the cascaded lattice Boltzmann method", *Physical Review E*, vol. 78, no. 1, p. 016 701, Jul. 2008.

[54] G. Gruszczyński, T. Mitchell, C. Leonardi, Ł. Łaniewski-Wołłk, and T. Barber, "A cascaded phase-field lattice Boltzmann model for the simulation of incompressible, immiscible fluids with high density contrast", *Computers and Mathematics with Applications*, vol. 79, no. 4, pp. 1049–1071, Feb. 2020.

[55] M. Geier, S. Lenz, M. Schönherr, and M. Krafczyk, "Under-resolved and large eddy simulations of a decaying Taylor–Green vortex with the cumulant lattice Boltzmann method", *Theoretical and Computational Fluid Dynamics*, vol. 35, no. 2, pp. 169–208, Nov. 2020.

[56] S. Simonis, M. Haussmann, L. Kronberg, W. Dörfler, and M. J. Krause, "Linear and brute force stability of orthogonal moment multiple-relaxation-time lattice Boltzmann methods applied to homogeneous isotropic turbulence", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2208, p. 20 200 405, Aug. 2021.

[57] I. Ginzburg, F. Verhaeghe, and D. d'Humières, "Two-Relaxation-Time Lattice Boltzmann Scheme: About Parametrization, Velocity, Pressure and Mixed Boundary Conditions", *Communications in Computational Physics*, vol. 3, no. 2, pp. 427–478, 2008.

[58] I. Ginzburg, F. Verhaeghe, and D. d'Humières, "Study of Simple Hydrodynamic Solutions with the Two-Relaxation-Times Lattice Boltzmann Scheme", *Communications in Computational Physics*, vol. 3, no. 3, pp. 519–581, 2008.

[59] M. Geier, A. Pasquali, and M. Schönherr, "Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part I: Derivation and validation", *Journal of Computational Physics*, vol. 348, pp. 862–888, Nov. 2017.

[60] M. Geier, A. Pasquali, and M. Schönherr, "Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part II: Application to flow around a sphere at drag crisis", *Journal of Computational Physics*, vol. 348, pp. 889–898, Nov. 2017.

[61] M. Gehrke and T. Rung, "Periodic hill flow simulations with a parameterized cumulant lattice Boltzmann method", *International Journal for Numerical Methods in Fluids*, vol. 94, no. 8, pp. 1111–1154, Apr. 2022.

[62] J. Latt and B. Chopard, "Lattice Boltzmann method with regularized pre-collision distribution functions", *Mathematics and Computers in Simulation*, vol. 72, no. 2–6, pp. 165–168, Sep. 2006.

[63] C. Coreixas, G. Wissocq, G. Puigt, J.-F. Boussuge, and P. Sagaut, "Recursive regularization step for high-order lattice Boltzmann methods", *Physical Review E*, vol. 96, no. 3, p. 033 306, Sep. 2017.

[64] J. Jacob, O. Malaspinas, and P. Sagaut, "A new hybrid recursive regularised Bhatnagar–Gross–Krook collision model for Lattice Boltzmann method-based large eddy simulation", *Journal of Turbulence*, vol. 19, no. 11–12, pp. 1051–1076, Nov. 2018.

[65] G. G. Spinelli, T. Horstmann, K. Masilamani, M. M. Soni, H. Klimach, A. Stück, and S. Roller, "HPC performance study of different collision models using the Lattice Boltzmann solver Musubi", *Computers & Fluids*, vol. 255, p. 105 833, Apr. 2023.

[66] I. V. Karlin, A. N. Gorban, S. Succi, and V. Boffi, "Maximum Entropy Principle for Lattice Kinetic Equations", *Physical Review Letters*, vol. 81, no. 1, pp. 6–9, Jul. 1998.

[67] B. M. Boghosian, J. Yepez, P. V. Coveney, and A. Wager, "Entropic lattice Boltzmann methods", *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 457, no. 2007, pp. 717–766, Mar. 2001.

[68] O. Malaspinas, M. Deville, and B. Chopard, "Towards a physical interpretation of the entropic lattice Boltzmann method", *Physical Review E*, vol. 78, no. 6, p. 066 705, Dec. 2008.

[69] I. V. Karlin, F. Bösch, and S. S. Chikatamarla, "Gibbs' principle for the lattice-kinetic theory of fluid dynamics", *Physical Review E*, vol. 90, no. 3, p. 031 302, Sep. 2014.

[70] F. Bösch, S. S. Chikatamarla, and I. V. Karlin, "Entropic multirelaxation lattice Boltzmann models for turbulent flows", *Physical Review E*, vol. 92, no. 4, p. 043 309, Oct. 2015.

[71] A. J. C. Ladd, "Numerical simulations of particulate suspensions via a discretized Boltzmann equation. part 1. theoretical foundation", *Journal of Fluid Mechanics*, vol. 271, pp. 285–309, Jul. 1994.

[72] I. Ginzbourg and P. M. Adler, "Boundary flow condition analysis for the three-dimensional lattice Boltzmann model", *Journal de Physique II*, vol. 4, no. 2, pp. 191–214, Feb. 1994.

[73] I. Ginzbourg and D. d'Humières, "Local second-order boundary methods for lattice Boltzmann models", *Journal of Statistical Physics*, vol. 84, no. 5–6, pp. 927–971, Sep. 1996.

[74] S. Khirevich, I. Ginzburg, and U. Tallarek, "Coarse- and fine-grid numerical behavior of MRT/TRT lattice-Boltzmann schemes in regular and random sphere packings", *Journal of Computational Physics*, vol. 281, pp. 708–742, Jan. 2015.

[75] M. Bouzidi, M. Firdaouss, and P. Lallemand, "Momentum transfer of a Boltzmann-lattice fluid with boundaries", *Physics of Fluids*, vol. 13, no. 11, pp. 3452–3459, Nov. 2001.

[76] Marson, Francesco, "Directional lattice Boltzmann boundary conditions", en, Ph.D. dissertation, 2022.

[77] I. Ginzburg, "Generic boundary conditions for lattice Boltzmann models and their application to advection and anisotropic dispersion equations", *Advances in Water Resources*, vol. 28, no. 11, pp. 1196–1216, Nov. 2005.

[78] R. Cornubert, D. d'Humières, and D. Levermore, "A Knudsen layer theory for lattice gases", *Physica D: Nonlinear Phenomena*, vol. 47, no. 1–2, pp. 241–259, Jan. 1991.

[79] T. Mitchell, "Development of a multiphase lattice Boltzmann model for high-density and viscosity ratio flows in unconventional gas wells", Ph.D. dissertation, University of Queensland Library, 2019.

[80] Z. Guo, C. Zheng, and B. Shi, "Discrete lattice effects on the forcing term in the lattice Boltzmann method", *Physical Review E*, vol. 65, no. 4, p. 046 308, Apr. 2002.

[81] G. Silva and V. Semiao, "First- and second-order forcing expansions in a lattice Boltzmann method reproducing isothermal hydrodynamics in artificial compressibility form", *Journal of Fluid Mechanics*, vol. 698, pp. 282–303, Apr. 2012.

[82] B. Postma and G. Silva, "Force methods for the two-relaxation-times lattice Boltzmann", *Physical Review E*, vol. 102, no. 6, p. 063 307, Dec. 2020.

[83] R. Mei, L.-S. Luo, P. Lallemand, and D. d'Humières, "Consistent initial conditions for lattice Boltzmann simulations", *Computers & Fluids*, vol. 35, no. 8–9, pp. 855–862, Sep. 2006.

[84] C. Körner, M. Thies, T. Hofmann, N. Thürey, and U. Rüde, "Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming", *Journal of Statistical Physics*, vol. 121, no. 1–2, pp. 179–196, Oct. 2005.

[85] T. Nils, P. T, and R. U, "Hybrid Parallelization Techniques for Lattice Boltzmann Free Surface Flows", in *Parallel Computational Fluid Dynamics 2007*. Springer Berlin Heidelberg, Feb. 2008, pp. 179–186.

[86] S. Bogner, R. Ammer, and U. Rüde, "Boundary conditions for free interfaces with the lattice Boltzmann method", *Journal of Computational Physics*, vol. 297, pp. 1–12, Sep. 2015.

[87] S. Bogner, U. Rüde, and J. Harting, "Curvature estimation from a volume-of-fluid indicator function for the simulation of surface tension and wetting with a free-surface lattice Boltzmann method", *Physical Review E*, vol. 93, no. 4, p. 043 302, Apr. 2016.

[88] C. Schwarzmeier and U. Rüde, "Analysis and comparison of boundary condition variants in the free-surface lattice Boltzmann method", *International Journal for Numerical Methods in Fluids*, vol. 95, no. 5, pp. 820–850, Jan. 2023.

[89] R. Scardovelli and S. Zaleski, "DIRECT NUMERICAL SIMULATION OF FREE-SURFACE AND INTERFACIAL FLOW", *Annual Review of Fluid Mechanics*, vol. 31, no. 1, pp. 567–603, Jan. 1999.

[90] A. Safi, "Efficient computations for multiphase flow problems using coupled lattice Boltzmann-level set methods", Ph.D. dissertation, Technische Universität Dortmund, 2016.

[91] C. Schwarzmeier and U. Rüde, "Comparison of refilling schemes in the free-surface lattice Boltzmann method", *AIP Advances*, vol. 12, no. 11, Nov. 2022.

[92] H. Huang, M. C. Sukop, and X.-Y. Lu, *Multiphase Lattice Boltzmann Methods: Theory and Application*. Wiley, Jun. 2015.

[93] A. K. Gunstensen, D. H. Rothman, S. Zaleski, and G. Zanetti, "Lattice Boltzmann model of immiscible fluids", *Physical Review A*, vol. 43, no. 8, pp. 4320–4327, Apr. 1991.

[94] M. Latva-Kokko and D. H. Rothman, "Diffusion properties of gradient-based lattice Boltzmann models of immiscible fluids", *Physical Review E*, vol. 71, no. 5, p. 056 702, May 2005.

[95] T. Reis and T. N. Phillips, "Lattice Boltzmann model for simulating immiscible two-phase flows", *Journal of Physics A: Mathematical and Theoretical*, vol. 40, no. 14, pp. 4033–4053, Mar. 2007.

[96] T. Lafarge, P. Boivin, N. Odier, and B. Cuenot, "Improved color-gradient method for lattice Boltzmann modeling of two-phase flows", *Physics of Fluids*, vol. 33, no. 8, p. 082 110, Aug. 2021.

[97] X. Shan and H. Chen, "Lattice Boltzmann model for simulating flows with multiple phases and components", *Physical Review E*, vol. 47, no. 3, pp. 1815–1819, Mar. 1993.

[98] X. Shan and H. Chen, "Simulation of nonideal gases and liquid-gas phase transitions by the lattice Boltzmann equation", *Physical Review E*, vol. 49, no. 4, pp. 2941–2948, Apr. 1994.

[99] D. Lycett-Brown and K. H. Luo, "Improved forcing scheme in pseudopotential lattice Boltzmann methods for multiphase flow at arbitrarily high density ratios", *Physical Review E*, vol. 91, no. 2, p. 023 305, Feb. 2015.

[100] P. Yuan and L. Schaefer, "Equations of state in a lattice Boltzmann model", *Physics of Fluids*, vol. 18, no. 4, Apr. 2006.

[101] J. Zhang and F. Tian, "A bottom-up approach to non-ideal fluids in the lattice Boltzmann method", *EPL (Europhysics Letters)*, vol. 81, no. 6, p. 66 005, Feb. 2008.

[102] M. R. Swift, W. R. Osborn, and J. M. Yeomans, "Lattice Boltzmann Simulation of Nonideal Fluids", *Physical Review Letters*, vol. 75, no. 5, pp. 830–833, Jul. 1995.

[103] M. R. Swift, E. Orlandini, W. R. Osborn, and J. M. Yeomans, "Lattice Boltzmann simulations of liquid-gas and binary fluid systems", *Physical Review E*, vol. 54, no. 5, pp. 5041–5052, Nov. 1996.

[104] D. J. Holdych, D. Rovas, J. G. Georgiadis, and R. O. Buckius, "An Improved Hydrodynamics Formulation for Multiphase Flow Lattice-Boltzmann Models", *International Journal of Modern Physics C*, vol. 09, no. 08, pp. 1393–1404, Dec. 1998.

[105] H. Zheng, C. Shu, and Y. Chew, "A lattice Boltzmann model for multiphase flows with large density ratio", *Journal of Computational Physics*, vol. 218, no. 1, pp. 353–371, Oct. 2006.

[106] J. W. Cahn and J. E. Hilliard, "Free Energy of a Nonuniform System. I. Interfacial Free Energy", *The Journal of Chemical Physics*, vol. 28, no. 2, pp. 258–267, Feb. 1958.

[107] M. Geier, A. Fakhari, and T. Lee, "Conservative phase-field lattice Boltzmann model for interface tracking equation", *Physical Review E*, vol. 91, no. 6, p. 063 309, Jun. 2015.

[108] C. Zhang, Z. Guo, and H. Liang, "High-order lattice-Boltzmann model for the Cahn-Hilliard equation", *Physical Review E*, vol. 99, no. 4, p. 043 310, Apr. 2019.

[109] X. He, S. Chen, and R. Zhang, "A Lattice Boltzmann Scheme for Incompressible Multiphase Flow and Its Application in Simulation of Rayleigh–Taylor Instability", *Journal of Computational Physics*, vol. 152, no. 2, pp. 642–663, Jul. 1999.

[110] X. He, R. Zhang, S. Chen, and G. D. Doolen, "On the three-dimensional Rayleigh–Taylor instability", *Physics of Fluids*, vol. 11, no. 5, pp. 1143–1152, May 1999.

[111] T. Lee and C.-L. Lin, "A stable discretization of the lattice Boltzmann equation for simulation of incompressible two-phase flows at high density ratio", *Journal of Computational Physics*, vol. 206, no. 1, pp. 16–47, Jun. 2005.

[112] P.-H. Chiu and Y.-T. Lin, "A conservative phase field method for solving incompressible two-phase flows", *Journal of Computational Physics*, vol. 230, no. 1, pp. 185–204, Jan. 2011.

[113] A. Fakhari, M. Geier, and D. Bolster, "A simple phase-field model for interface tracking in three dimensions", *Computers and Mathematics with Applications*, vol. 78, no. 4, pp. 1154–1165, Aug. 2019.

[114] A. Kumar, "Isotropic finite-differences", *Journal of Computational Physics*, vol. 201, no. 1, pp. 109–118, Nov. 2004.

[115] F. Ren, B. Song, M. C. Sukop, and H. Hu, "Improved lattice Boltzmann modeling of binary flow based on the conservative Allen-Cahn equation", *Physical Review E*, vol. 94, no. 2, p. 023 311, Aug. 2016.

[116] H. Liang, J. Xu, J. Chen, H. Wang, Z. Chai, and B. Shi, "Phase-field-based lattice Boltzmann modeling of large-density-ratio two-phase flows", *Physical Review E*, vol. 97, no. 3, p. 033 309, Mar. 2018.

[117] T. Mitchell, C. Leonardi, and A. Fakhari, "Development of a three-dimensional phase-field lattice Boltzmann method for the study of immiscible fluids at high density ratios", *International Journal of Multiphase Flow*, vol. 107, pp. 1–15, Oct. 2018.

[118] T. Mitchell and C. Leonardi, "Development of closure relations for the motion of Taylor bubbles in vertical and inclined annular pipes using high-fidelity numerical modeling", *Physics of Fluids*, vol. 32, no. 6, Jun. 2020.

[119] A. Fakhari and D. Bolster, "Diffuse interface modeling of three-phase contact line dynamics on curved boundaries: a lattice Boltzmann model for large density and viscosity ratios", *Journal of Computational Physics*, vol. 334, pp. 620–638, Apr. 2017.

[120] A. Fakhari, Y. Li, D. Bolster, and K. T. Christensen, "A phase-field lattice Boltzmann model for simulating multiphase flows in porous media: Application and comparison to experiments of CO2 sequestration at pore scale", *Advances in Water Resources*, vol. 114, pp. 119–134, Apr. 2018.

[121] K. T. Kotz, K. A. Noble, and G. W. Faris, "Optical microfluidics", *Applied Physics Letters*, vol. 85, no. 13, pp. 2658–2660, Sep. 2004.

[122] L. Yue, Z. Chai, H. Wang, and B. Shi, "Improved phase-field-based lattice Boltzmann method for thermocapillary flow", *Physical Review E*, vol. 105, no. 1, p. 015 314, Jan. 2022.

[123] Y. Abe, A. Iwasaki, and K. Tanaka, "Microgravity experiments on phase change of self-rewetting fluids", *Annals of the New York Academy of Sciences*, vol. 1027, no. 1, pp. 269–285, Nov. 2004.

[124] B. Elbousefi, W. Schupbach, K. N. Premnath, and S. W. Welch, "Thermocapillary convection in superimposed layers of self-rewetting fluids: analytical and lattice Boltzmann computational study", *International Journal of Heat and Mass Transfer*, vol. 208, p. 124 049, Jul. 2023.

[125] H. Liu, Y. Zhang, and A. J. Valocchi, "Modeling and simulation of thermocapillary flows using lattice Boltzmann method", *Journal of Computational Physics*, vol. 231, no. 12, pp. 4433–4453, Jun. 2012.

[126] H. Liu, A. J. Valocchi, Y. Zhang, and Q. Kang, "Phase-field-based lattice Boltzmann finite-difference model for simulating thermocapillary flows", *Physical Review E*, vol. 87, no. 1, p. 013 010, Jan. 2013.

[127] H. Liu, A. J. Valocchi, Y. Zhang, and Q. Kang, "Lattice Boltzmann phase-field modeling of thermocapillary flows in a confined microchannel", *Journal of Computational Physics*, vol. 256, pp. 334–356, Jan. 2014.

[128] T. R. Mitchell, M. Majidi, M. H. Rahimian, and C. R. Leonardi, "Computational modeling of three-dimensional thermocapillary flow of recalcitrant bubbles using a coupled lattice Boltzmann-finite difference method", *Physics of Fluids*, vol. 33, no. 3, Mar. 2021.

[129] F. L. Bauer and H. Wössner, "The "Plankalkül" of Konrad Zuse: a forerunner of today's programming languages", *Communications of the ACM*, vol. 15, no. 7, pp. 678–685, Jul. 1972.

[130] M. Fowler, *Domain-specific languages* (The Addison-Wesley signature series). Upper Saddle River, N.J.: Addison-Wesley, 2011, Description based on print version record. - "A Martin Fowler signature book."–Cover. - Includes bibliographical references and index.

[131] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "SymPy: symbolic computing in Python", *PeerJ Computer Science*, vol. 3, e103, Jan. 2017.

[132] A. Klöckner, "Loo.py: transformation-based code generation for GPUs and CPUs", in *Proceedings of ARRAY '14: ACM SIGPLAN Workshop on Libraries, Languages, and Compilers for Array Programming*, ACM, Jun. 9, 2014.

[133] S. Verdoolaege, "isl: An Integer Set Library for the Polyhedral Model", in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 299–302.

[134] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation", *Parallel Computing*, vol. 38, no. 3, pp. 157–174, Mar. 2012.

[135] C. Lengauer, S. Apel, M. Bolten, A. Größlinger, F. Hannig, H. Köstler, U. Rüde, J. Teich, A. Grebhahn, S. Kronawitter, S. Kuckuk, H. Rittich, and C. Schmitt, "ExaStencils: Advanced Stencil-Code Engineering", in *Euro-Par 2014: Parallel Processing Workshops*. Springer International Publishing, 2014, pp. 553–564.

[136] S. Faghih-Naini and V. Aizinger, "p-adaptive discontinuous Galerkin method for the shallow water equations with a parameter-free error indicator", *GEM - International Journal on Geomathematics*, vol. 13, no. 1, Oct. 2022.

[137] H. Martiros, A. Miller, N. Bucki, B. Solliday, R. Kennedy, J. Zhu, T. Dang, D. Pattison, H. Zheng, T. Tomic, P. Henry, G. Cross, J. VanderMey, A. Sun, S. Wang, and K. Holtz, "SymForce: Symbolic Computation and Code Generation for Robotics", in *Robotics: Science and Systems XVIII*, ser. RSS2022, Robotics: Science and Systems Foundation, Jun. 2022.

[138] C. Yount, J. Tobin, A. Breuer, and A. Duran, "YASK—Yet Another Stencil Kernel: A Framework for HPC Stencil Code-Generation and Tuning", in *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*, IEEE, Nov. 2016.

[139] J. Hammer, J. Eitzinger, G. Hager, and G. Wellein, "Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels", in *Tools for High Performance Computing 2016*. Springer International Publishing, 2017, pp. 1–22.

[140] C. Alappat, G. Hager, O. Schenk, and G. Wellein, "Level-Based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication", *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 581–597, Feb. 2023.

[141] P. A. Skordos, "Initial and boundary conditions for the lattice Boltzmann method", *Physical Review E*, vol. 48, no. 6, pp. 4823–4842, Dec. 1993.

[142] A. De Rosis, "Nonorthogonal central-moments-based lattice Boltzmann scheme in three dimensions", *Physical Review E*, vol. 95, no. 1, p. 013 310, Jan. 2017.

[143] N. Kohl, J. Hötzer, F. Schornbaum, M. Bauer, C. Godenschwager, H. Köstler, B. Nestler, and U. Rüde, "A scalable and extensible checkpointing scheme for massively parallel simulations", *The International Journal of High Performance Computing Applications*, vol. 33, no. 4, pp. 571–589, May 2018.

[144] M. Bauer, F. Schornbaum, C. Godenschwager, M. Markl, D. Anderl, H. Köstler, and U. Rüde, "A Python extension for the massively parallel multiphysics simulation framework WALBERLA", *International Journal of Parallel, Emergent and Distributed Systems*, vol. 31, no. 6, pp. 529–542, Dec. 2015.

[145] M. J. Krause, A. Kummerländer, S. J. Avis, H. Kusumaatmaja, D. Dapelo, F. Klemens, M. Gaedtke, N. Hafen, A. Mink, R. Trunk, J. E. Marquardt, M.-L. Maier, M. Haussmann, and S. Simonis, "OpenLB—Open source lattice Boltzmann code", *Computers & Mathematics with Applications*, vol. 81, pp. 258–288, Jan. 2021.

[146] A. Kummerländer, T. Bingert, F. Bukreev, L. E. Czelusniak, D. Dapelo, N. Hafen, M. Heinzelmann, S. Ito, J. Jeßberger, H. Kusumaatmaja, J. E. Marquardt, M. Rennick, T. Pertzel, F. Prinz, M. Sadric, M. Schecher, S. Simonis, P. Sitter, D. Teutscher, M. Zhong, and M. J. Krause, *OpenLB Release 1.7: Open Source Lattice Boltzmann Code*, en, 2024.

[147] J. Latt, O. Malaspinas, D. Kontaxakis, A. Parmigiani, D. Lagrava, F. Brogi, M. B. Belgacem, Y. Thorimbert, S. Leclaire, S. Li, F. Marson, J. Lemus, C. Kotsalos, R. Conradin, C. Coreixas, R. Petkantchin, F. Raynaud, J. Beny, and B. Chopard, "Palabos: Parallel Lattice Boltzmann Solver", *Computers & Mathematics with Applications*, vol. 81, pp. 334–350, Jan. 2021.

[148] S. Peters, K. Kutscher, M. Schönherr, M. Geier, H. Alihussein, A. Wellmann, and H. Korb, *VirtualFluids*, 2024.

[149] M. Geier, A. Greiner, and J. G. Korvink, "Bubble functions for the lattice Boltzmann method and their application to grid refinement", *The European Physical Journal Special Topics*, vol. 171, no. 1, pp. 173–179, Apr. 2009.

[150] J. Qi, H. Klimach, and S. Roller, "Implementation of the compact interpolation within the octree based Lattice Boltzmann solver Musubi", *Computers & Mathematics with Applications*, vol. 78, no. 4, pp. 1131–1141, Aug. 2019.

[151] M. Lehmann, M. J. Krause, G. Amati, M. Sega, J. Harting, and S. Gekle, "Accuracy and performance of the lattice Boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats", *Physical Review E*, vol. 106, no. 1, p. 015 308, Jul. 2022.

[152] Łukasz Łaniewski-Wołłk, Michał Dzikowski, T. Mitchell, D. Sashko, Ggruszczynski, Paweł Obrępalski, Mrutkowski-Aero, W. Regulski, Bhill23, TGajek, JonMcCullough, Corneels De Waard, and Lanwatch, *CFD-GO/TCLB: Version 6.7*, 2023.

[153] Ł. Łaniewski-Wołłk and J. Rokicki, "Adjoint Lattice Boltzmann for topology optimization on multi-GPU architecture", *Computers & Mathematics with Applications*, vol. 71, no. 3, pp. 833–848, Feb. 2016.

[154] A. Sengissen, J.-C. Giret, C. Coreixas, and J.-F. Boussuge, "Simulations of LAGOON landing-gear noise using Lattice Boltzmann Solver", in *21st AIAA/CEAS Aeroacoustics Conference*, American Institute of Aeronautics and Astronautics, Jun. 2015.

[155] A. Pasquali, "Enabling the cumulant lattice Boltzmann methodfor complex CFD engineering problems", Ph.D. dissertation, Technische Universität Carolo-Wilhelmina zu Braunschweig, 2016.

[156] C. Feichtinger, "Design and Performance Evaluation of a Software Framework for Multi-Physics Simulations on Heterogeneous Supercomputers", Ph.D. dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2012.

[157] S. Eibl and U. Rüde, "A Modular and Extensible Software Architecture for Particle Dynamics", in *8th international conference on discrete element methods (DEM8)*, 2019.

[158] C. Schwarzmeier, C. Rettinger, S. Kemmler, J. Plewinski, F. Núñez-González, H. Köstler, U. Rüde, and B. Vowinckel, "Particle-resolved simulation of antidunes in free-surface flows", *Journal of Fluid Mechanics*, vol. 961, Apr. 2023.

[159] O. Filippova and D. Hänel, "Grid Refinement for Lattice-BGK Models", *Journal of Computational Physics*, vol. 147, no. 1, pp. 219–228, Nov. 1998.

[160] M. Rohde, D. Kandhai, J. J. Derksen, and H. E. A. van den Akker, "A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes", *International Journal for Numerical Methods in Fluids*, vol. 51, no. 4, pp. 439–468, Dec. 2005.

[161] A. Schukmann, A. Schneider, V. Haas, and M. Böhle, "Analysis of Hierarchical Grid Refinement Techniques for the Lattice Boltzmann Method by Numerical Experiments", *Fluids*, vol. 8, no. 3, p. 103, Mar. 2023.

[162] C.-L. Lin and Y. G. Lai, "Lattice Boltzmann method on composite grids", *Physical Review E*, vol. 62, no. 2, pp. 2219–2225, Aug. 2000.

[163] A. Dupuis and B. Chopard, "Theory and applications of an alternative lattice Boltzmann grid refinement algorithm", *Physical Review E*, vol. 67, no. 6, p. 066 707, Jun. 2003.

[164] D. Yu, R. Mei, and W. Shyy, "A multi-block lattice Boltzmann method for viscous fluid flows", *International Journal for Numerical Methods in Fluids*, vol. 39, no. 2, pp. 99–120, Apr. 2002.

[165] C. Shannon, "Communication in the Presence of Noise", *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949.

[166] D. Lagrava, O. Malaspinas, J. Latt, and B. Chopard, "Advances in multi-domain lattice Boltzmann grid refinement", *Journal of Computational Physics*, vol. 231, no. 14, pp. 4808–4822, May 2012.

[167] P. Quéméré, P. Sagaut, and V. Couailler, "A new multi-domain/multi-resolution method for large-eddy simulation", *International Journal for Numerical Methods in Fluids*, vol. 36, no. 4, pp. 391–416, Jun. 2001.

[168] H. Touil, D. Ricot, and E. Lévêque, "Direct and large-eddy simulation of turbulent flows on composite multi-resolution grids by the lattice Boltzmann method", *Journal of Computational Physics*, vol. 256, pp. 220–233, Jan. 2014.

[169] T. Astoul, G. Wissocq, J.-F. Boussuge, A. Sengissen, and P. Sagaut, "Lattice Boltzmann method for computational aeroacoustics on non-uniform meshes: a direct grid coupling approach", *Journal of Computational Physics*, vol. 447, p. 110 667, Dec. 2021.

[170] H. Chen, O. Filippova, J. Hoch, K. Molvig, R. Shock, C. Teixeira, and R. Zhang, "Grid refinement in lattice Boltzmann methods based on volumetric formulation", *Physica A: Statistical Mechanics and its Applications*, vol. 362, no. 1, pp. 158–167, Mar. 2006.

[171] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger, and M. Krafczyk, "Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs", *Computers & Mathematics with Applications*, vol. 61, no. 12, pp. 3730–3743, Jun. 2011.

[172] M. Wittmann, V. Haag, T. Zeiser, H. Köstler, and G. Wellein, "Lattice Boltzmann benchmark kernels as a testbed for performance analysis", *Computers & Fluids*, vol. 172, pp. 582–592, Aug. 2018.

[173] G. Eiffel, *Nouvelles recherches sur la résistance de l'air et l'aviation faites au laboratoire d'Auteuil*. H. Dunod et E. Pinat, 1914.

[174] B. Chanetz, "A century of wind tunnels since Eiffel", *Comptes Rendus. Mécanique*, vol. 345, no. 8, pp. 581–594, Jul. 2017.

[175] E. Achenbach, "Experiments on the flow past spheres at very high Reynolds numbers", *Journal of Fluid Mechanics*, vol. 54, no. 3, pp. 565–575, Aug. 1972.

[176] L. Werner, C. Rettinger, and U. Rüde, "Coupling fully resolved light particles with the lattice Boltzmann method on adaptively refined grids", *International Journal for Numerical Methods in Fluids*, vol. 93, no. 11, pp. 3280–3303, Aug. 2021.

[177] A. S. Ambekar, C. Schwarzmeier, U. Rüde, and V. V. Buwa, "Particle-resolved turbulent flow in a packed bed: RANS, LES, and DNS simulations", *AIChE Journal*, vol. 69, no. 1, Feb. 2022.

[178] R. Mei, D. Yu, W. Shyy, and L.-S. Luo, "Force evaluation in the lattice Boltzmann method involving curved geometry", *Physical Review E*, vol. 65, no. 4, p. 041 203, Apr. 2002.

[179] L. Prandtl, *Abriß der Strömungslehre*. Göttingen University Press, 2017.

[180] F. A. Morrison, *An introduction to fluid mechanics*. Cambridge: Cambridge University Press, 2013, 1927 pp., Title from publisher's bibliographic system (viewed on 18 Jul 2016).

[181] J. Almedeij, "Drag coefficient of flow around a sphere: matching asymptotically the wide trend", *Powder Technology*, vol. 186, no. 3, pp. 218–223, Sep. 2008.

[182] T. Astoul, G. Wissocq, J.-F. Boussuge, A. Sengissen, and P. Sagaut, "Analysis and reduction of spurious noise generated at grid refinement interfaces with the lattice Boltzmann method", *Journal of Computational Physics*, vol. 418, p. 109 645, Oct. 2020.

[183] C. Feuchter, "Direct aeroacoustic simulation with a cumulant Lattice-Boltzmann model", *Computers & Fluids*, vol. 224, p. 104 970, Jun. 2021.

[184] B. Wu, M. Firouzi, T. Mitchell, T. E. Rufford, C. Leonardi, and B. Towler, "A critical review of flow maps for gas-liquid flows in vertical pipes and annuli", *Chemical Engineering Journal*, vol. 326, pp. 350–377, Oct. 2017.

[185] E. M. A. Frederix, E. M. J. Komen, I. Tiselj, and B. Mikuž, "LES of Turbulent Co-current Taylor Bubble Flow", *Flow, Turbulence and Combustion*, vol. 105, no. 2, pp. 471–495, Mar. 2020.

[186] A. Tomiyama, Y. Nakahara, Y. Adachi, and S. Hosokawa, "Shapes and Rising Velocities of Single Bubbles rising through an Inner Subchannel", *Journal of Nuclear Science and Technology*, vol. 40, no. 3, pp. 136–142, Mar. 2003.

[187] E. Gutiérrez, N. Balcázar, E. Bartrons, and J. Rigola, "Numerical study of Taylor bubbles rising in a stagnant liquid using a level-set/moving-mesh method", *Chemical Engineering Science*, vol. 164, pp. 158–177, Jun. 2017.

[188] G. Das, P. Das, N. Purohit, and A. Mitra, "Rise velocity of a Taylor bubble through concentric annulus", *Chemical Engineering Science*, vol. 53, no. 5, pp. 977–993, Feb. 1998.

[189] F. Viana, P. Pardo, R. Yanex, J. L. Trallero, and D. D. Joseph, "Universal correlation for the rise velocity of long gas bubbles in round pipes", *Journal of Fluid Mechanics*, vol. 494, pp. 379–398, Nov. 2003.

[190] V. Hernández-Pérez, L. A. Abdulkareem, and B. J. Azzopardi, "Effects of physical properties on the behaviour of Taylor bubbles", in *WIT Transactions on Engineering Sciences*, ser. MPF09, WIT Press, Jun. 2009.

[191] J. Moreiras, E. Pereyra, C. Sarica, and C. F. Torres, "Unified drift velocity closure relationship for large bubbles rising in stagnant viscous fluids in pipes", *Journal of Petroleum Science and Engineering*, vol. 124, pp. 359–366, Dec. 2014.

[192] E. Lizarraga-Garcia, J. Buongiorno, E. Al-Safran, and D. Lakehal, "A broadly-applicable unified closure relation for Taylor bubble rise velocity in pipes with stagnant liquid", *International Journal of Multiphase Flow*, vol. 89, pp. 345–358, Mar. 2017.

[193] E. Z. Massoud, Q. Xiao, H. A. El-Gamal, and M. A. Teamah, "Numerical study of an individual Taylor bubble rising through stagnant liquids under laminar flow regime", *Ocean Engineering*, vol. 162, pp. 117–137, Aug. 2018.

[194] J. Bugg and G. Saad, "The velocity field around a Taylor bubble rising in a stagnant viscous fluid: numerical and experimental results", *International Journal of Multiphase Flow*, vol. 28, no. 5, pp. 791–803, May 2002.

[195] T. Taha and Z. Cui, "CFD modelling of slug flow in vertical tubes", *Chemical Engineering Science*, vol. 61, no. 2, pp. 676–687, Jan. 2006.

[196] M. C. F. Silva, J. B. L. M. Campos, and J. D. P. Araújo, "Mass transfer from a soluble Taylor bubble to the surrounding flowing liquid in a vertical macro tube - A numerical approach", *Chemical Engineering Research and Design*, vol. 144, pp. 47–62, Apr. 2019.

[197] D. Anderl, S. Bogner, C. Rauh, U. Rüde, and A. Delgado, "Free surface lattice Boltzmann with enhanced bubble model", *Computers & Mathematics with Applications*, vol. 67, no. 2, pp. 331–339, Feb. 2014.

[198] T. Mitchell and C. Leonardi, "On the rise characteristics of Taylor bubbles in annular piping", *International Journal of Multiphase Flow*, vol. 130, p. 103 376, Sep. 2020.

[199] W. Li, D. Liu, M. Desbrun, J. Huang, and X. Liu, "Kinetic-Based Multiphase Flow Simulation", *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 7, pp. 3318–3334, Jul. 2021.

[200] A. Savitzky and M. J. E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964.

[201] B. Pendse and A. Esmaeeli, "An analytical solution for thermocapillary-driven convection of superimposed fluids at zero Reynolds and Marangoni numbers", *International Journal of Thermal Sciences*, vol. 49, no. 7, pp. 1147–1155, Jul. 2010.

[202] C. N. Baroud, J.-P. Delville, F. Gallaire, and R. Wunenburger, "Thermocapillary valve for droplet production and sorting", *Physical Review E*, vol. 75, no. 4, p. 046 302, Apr. 2007.

[203] S. Sohrabi, N. kassir, and M. Keshavarz Moraveji, "Droplet microfluidics: fundamentals and its advanced applications", *RSC Advances*, vol. 10, no. 46, pp. 27 560–27 574, 2020.

[204] B. J. Won, W. Lee, and S. Song, "Estimation of the thermocapillary force and its applications to precise droplet control on a microfluidic chip", *Scientific Reports*, vol. 7, no. 1, Jun. 2017.

[205] M. Majidi, R. Haghani-Hassan-Abadi, and M.-H. Rahimian, "Single recalcitrant bubble simulation using a hybrid lattice Boltzmann finite difference model", *International Journal of Multiphase Flow*, vol. 127, p. 103 289, Jun. 2020.

[206] S. Kesselheim, A. Herten, K. Krajsek, J. Ebert, J. Jitsev, M. Cherti, M. Langguth, B. Gong, S. Stadtler, A. Mozaffari, G. Cavallaro, R. Sedona, A. Schug, A. Strube, R. Kamath, M. G. Schultz, M. Riedel, and T. Lippert, "JUWELS Booster – A Supercomputer for Large-Scale AI Research", in *High Performance Computing*. Springer International Publishing, 2021, pp. 453–468.

[207] A. Herten, B. von St. Vieth, D. Alvarez, E. Suarez, W. Frings, and T. Eickermann, "Reaching JUPITER: Mission Details and Launch Preparations", en, *16th JLESC Workshop*, 2024.

[208] NVIDIA, "NVIDIA GH200 Grace Hopper Superchip", NVIDIA, Tech. Rep., 2024.

[209] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures", *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[210] H. Stengel, J. Treibig, G. Hager, and G. Wellein, "Quantifying Performance Bottlenecks of Stencil Computations Using the Execution-Cache-Memory Model", in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS'15, ACM, Jun. 2015.

[211] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments", in *2010 39th International Conference on Parallel Processing Workshops*, IEEE, Sep. 2010.

[212] M. Wittmann, G. Hager, T. Zeiser, J. Treibig, and G. Wellein, "Chip-level and multi-node analysis of energy-optimized lattice Boltzmann CFD simulations", *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 2295–2315, May 2015.

[213] Z. Liu, X. Chu, X. Lv, H. Meng, H. Liu, G. Zhu, H. Fu, and G. Yang, "SunwayLB: Enabling Extreme-Scale Lattice Boltzmann Method Based Computing Fluid Dynamics Simulations on Advanced Heterogeneous Supercomputers", *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 2, pp. 324–337, Feb. 2024.

[214] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The Sunway TaihuLight supercomputer: system and applications", *Science China Information Sciences*, vol. 59, no. 7, Jun. 2016.

[215] J. Tölke and M. Krafczyk, "TeraFLOP computing on a desktop PC with GPUs for 3D CFD", *International Journal of Computational Fluid Dynamics*, vol. 22, no. 7, pp. 443–456, Jul. 2008.

[216] J. Habich, C. Feichtinger, H. Köstler, G. Hager, and G. Wellein, "Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results", *Computers & Fluids*, vol. 80, pp. 276–282, Jul. 2013.

[217] "IEEE Standard for Floating-Point Arithmetic", *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[218] Y. P. Sitompul and T. Aoki, "A filtered cumulant lattice Boltzmann method for violent two-phase flows", *Journal of Computational Physics*, vol. 390, pp. 93–120, Aug. 2019.