TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Numerische Mechanik

Performance Aspects of Incompressible Navier-Stokes Solvers: Lattice-Boltzmann Method vs. Finite Difference Method

Karl-Robert Klaus Wichmann

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzende:	Prof. Dr.-Ing. Birgit Vogel-Heuser
Prüfer der Dissertation:	
	1. Prof. Dr.-Ing. Wolfgang A. Wall
	2. Prof. Dr. rer. nat. Gerhard Wellein

Die Dissertation wurde am 19.04.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 15.03.2019 angenommen.

Abstract

The use of fluid simulations is an integral part of research and development. Nearly every branch of industry is using simulations to speed up development and reduce costs. Fast computational fluid dynamics (CFD) are critical and subject to much research. Many publications demonstrate the high performance for countless different methods. Mostly however, only high convergence orders or high update rates are shown. In this work the focus is on practical performance aspects, which means that the performance is gauged by the result quality per runtime. Furthermore CFD is often used to discover trends, as uncertainties in the input parameters inhibit exact predictions. In this context the comparisons are carried out with relatively coarse results and up to a few significant digits only.

The CFD in its entirety is orders of magnitude too large to be explored extensively. It is therefore that two methods with a reputation of high performance have been selected for the comparison process. Especially the lattice-Boltzmann method (LBM) is often praised for its performance and it is compared to an artificial compressibility finite difference method (FDM) with similar behavior. These methods were chosen not only for their in general high CFD performance, but also for a second aspect that is investigated. Due to the similarity it can be established whether it is really the fundamentally different lattice-Boltzmann approach that is responsible for the high performance.

The comparison process is divided into the three categories of the method configuration, the code optimization, and the test case setup and evaluation. These categories minimize the interdependence, which naturally arise when establishing absolute CFD performance figures. The methods that are employed are a two relaxation time LBM and an artificial compressibility FDM. This particular choice of LBM is very simple and fast, yet also avoids some of the known LBM difficulties. The FDM is set up to closely resemble the LBM's behavior to bring out the differences of the underlying approaches. Both implementations are extensively tuned, to ensure they operate at optimal performance. The optimizations are verified with the help of different performance models. As these models exhibit significant errors when applied to highly tuned implementations, an investigation is carried out to determine the origins of these discrepancies.

The most important aspect of the performance comparison are the test cases. A duct flow, lid driven cavity, and steady and unsteady flows past a cylinder with a square cross section are used. These offer increasing flow complexity and a gradual change from a direct comparison to analytical solutions to the Navier-Stokes equations to complex quantities, such as the time-variant lift coefficient. For all these test both methods exhibited a close performance level and a very similar behavior in general. A tendency that can be observed throughout all examples is that the LBM has small performance advantage for coarser resolutions and larger errors, as well as with the indirect quantities of interest. In contrast the FDM is faster for stricter errors and higher resolutions and more importantly can reach smaller errors when compared directly to a solution of the Navier-Stokes equations. Apart from this performance conclusion some secondary factors that may be relevant to a method selection are provided.

Zusammenfassung

Heutzutage sind Fluidsimulationen fester Bestandteil der Forschung und Entwicklung und kaum ein Industriezweig kommt noch ohne Simulationen aus. Schnelle Simulationstechniken sind daher unabdingbar und der Gegenstand von vielen Forschungsarbeiten. Es gibt viele Veröffentlichungen die den verschiedensten Methoden hohe Leistungsfähigkeit bescheinigen, jedoch beschränken sich die meisten Arbeiten auf einzelne Aspekte, wie hohe Konvergenzordnungen oder einen hohen Durchsatz an Zeitschritten. Im Gegensatz dazu wird hier die Leistungsfähigkeit stattdessen in einer für die Praxis relevanten Art gemessen, in dem die tatsächlich erreichte Güte der Ergebnisse pro verstrichener Zeit betrachtet wird. Des Weiteren werden Fluidsimulationen häufig nur zum Bestimmen von Tendenzen herangezogen, da oftmals Unsicherheiten in den Parametern keine präzisen Ergebnisse zulassen. Daher werden in diesem Vergleich alle Resultate nur auf die ersten Nachkommastellen genau betrachtet.

Der vollständige Bereich der Fluidsimulationen umfasst selbstverständlich weit mehr Aspekte als behandelt werden können. Von daher werden hier ins Besondere zwei Methoden betrachtet, die den Ruf haben sehr Leistungsfähig zu sein. Insbesondere die Lattice-Boltzmann-Methode (LBM) als ein relativ neuer Ansatz wird häufig als besonders Leistungsfähig bezeichnet. Verglichen wird diese mit der Finite-Differenzen-Methode (FDM) in einer künstlichen kompressibilitäts Variante. Diese Wahl erlaubt nicht nur den Vergleich zweier sehr schneller Simulationsmethoden, sondern die Ähnlichkeit in ihrem Verhalten ermöglicht zudem noch Untersuchungen, ob es wirklich der Lattice-Boltzmann Ansatz ist, der dieser Methode zu hoher Leistungsfähigkeit verhilft.

Für den Vergleich werden drei Kategorien betrachtet, die die Abhängigkeiten untereinander möglichst gering halten. Diese Kategorien sind die Wahl der Methoden, die Optimierung der Implementierung und die Wahl der Beispiele und gemessenen Größen. Dennock unterliegen auch diese Kategorien einer Abhängigkeit, wenn das Gesamtthema schneller Löser für Fluidprobleme betrachtet wird.

Konkret handelt es sich bei den verglichenen Methoden um eine zwei Relaxationszeiten Lattice-Boltzmann-Methode, die trotz ihrer einfachen Struktur einige bekannte Probleme der LBM umgeht, sowie die Finite-Differenzen-Methode, welche mit ähnlichen Charakteristiken wie die LBM ausgestattet wurde. Beide Implementierungen sind umfangreich optimiert, so dass davon ausgegangen werden kann, dass beide Varianten bei ihrer vollen Leistungsfähigkeit sind. Die Optimierungen sind außerdem mit Hilfe von Performance Modellen verifiziert worden. Im Rahmen dessen wurde auch die Genauigkeit verschiedener Performance Modelle untersucht, da sie bei der Anwendung auf hoch optimierte Implementierungen erheblichen Abweichungen unterliegen können.

Der wichtigste Aspekt sind jedoch die Beispiele, anhand derer die Leistungsfähigkeit bestimmt wird. Konkret werden dazu eine Kanalströmung, eine Nischenströmung, sowie stationäre und instationäre Strömungen um einen Quader betrachtet. Diese bieten ein breites Spektrum an Herausforderungen, von einfachen Randbedingungen und Vergleich mit einer analytischen Lösung der Navier-Stokes Gleichungen, bis zur Auswertung von indirekten Größen wie Auftrieb

bei instationärem Verhalten. Im Rahmen dieser Untersuchungen hat sich wiederholt ein ähnliches Verhalten und eine insgesamt vergleichbare Geschwindigkeit beider Methoden gezeigt. Somit ist es nicht der Lattice-Boltzmann Ansatz, der die Leistungsfähigkeit bestimmt. Es sind jedoch bei allen Beispielen Tendenzen zu Tage getreten, die zeigen, dass die LBM bei größeren Fehlern und indirekten Messwerten einen Geschwindigkeitsvorsprung hat, wohingegen die FDM für kleinere Fehler und direktem Vergleich der primären Größen zu Lösungen der Navier-Stokes Gleichungen schneller ist. Zu dem reinen Geschwindigkeitsvergleich werden außerdem noch sekundäre Aspekte genannt, die bei der Wahl einer Methode berücksichtigt werden sollten.

Contents

Nomenclature

Abbreviations

ACM	Artificial compressibility method
AoS	array of structure
AoSoA	Array of Structures of Arrays
ARM	Advanced RISC machine
AV	Artificial viscosity
AVX	Advanced vector extensions
BB	Bounce-back
BE	Back end
BGK	Bhatnagar, Gross, Krook
BW	Bandwidth
ccNUMA	Cache coherent NUMA
CFD	Computational fluid dynamics
CL	Cache line
COD	Cluster-on-die
CS	Coordinate splitting
DNS	Direct numerical simulation
DOF	Degree of freedom
ECM	Execution-cache-memory
FDM	Finite difference method
FE	Front end
FLOP	Floating point operation
FMA	Fused-multiply add
IACA	Intel Architecture Code Analyzer
ILBDC	International Lattice Boltzmann Development Consortium
INSTR	Instruction
KNL	Knights Landing
LBGK	Lattice Bhatnagar-Gross-Krook
LBM	Lattice Boltzmann method
LDC	Lid driven cavity
LGCA	Lattice gas cellular automata
MCDRAM	Multi-channel dynamic RAM
MD	Molecular dynamics
MIC	Many Integrated Core
MM	Main memory
MRT	Multi-relaxation-time

NSE	Navier-Stokes equations
NUMA	Non-uniform memory access
OOO	Out-of-order
PAB	Pressure anti-bounce-back
PDE	Partial differential equation
QOI	Quantity of interest
QPI	QuickPath Interconnect
QWORD	Quadword
RAM	Random access memory
RFO	Read-for-ownership
RHS	Right hand side
RK	Runge-Kutta
SIMD	Single instruction multiple data
SoA	Structure of arrays
SRT	Single relaxation time
TLB	Translation lookaside buffer
TRT	Two-relaxation-time
TTS	Total time to solution

Dimensionless Numbers

Ma	Mach number
Re	Reynolds number
Re_{cell}	Cell Reynolds number
St	Strouhal number

Finite Difference Symbols

$\boldsymbol{f}_b$	Body force
Δt	Time step size
Δx	Grid spacing
$\boldsymbol{\epsilon}$	Rate of strain tensor
$\boldsymbol{\tau}$	Stress tensor
μ	Dynamic viscosity
ν	Kinematic viscosity
ρ	Density
$\boldsymbol{f}_s$	Surface force
$\boldsymbol{n}$	Surface normal
$\boldsymbol{R}$	Residual of momentum equation
$\boldsymbol{u}$	Velocity
c_s	Artificial speed of sound
c_{pres}	Pressure stabilization parameter
c_{vel}	Velocity stabilization parameter
D_p	Pressure stabilization term
D_u	Velocity stabilization term

p	Pressure
R	Residual of continuity equation

Lattice-Boltzmann Symbols

λ_e	Symmetric relaxation time
λ_o	Anti-symmetric relaxation time
Λ_{eo}	Relaxation time parameter
ν	Viscosity
ω	Relaxation time
ρ	Macroscopic density
σ	Differential collision cross section
$\boldsymbol{j}$	Macroscopic momentum
$\boldsymbol{K}$	External force
$\boldsymbol{p}$	Particle momentum
$\boldsymbol{q}$	Particle coordinate
$\boldsymbol{u}$	Macroscopic velocity
$\boldsymbol{u}$	Mean velocity
$\boldsymbol{v}$	Particle velocity
$\boldsymbol{x}$	Particle coordinate
c	Lattice velocity
c_s	Speed of sound
f	Density distribution function
$f^{(eq)}$	Equilibrium density distribution function
$f^{(M)}$	Local Maxwellian
J	Collision integral approximation
m	Particle mass
N	Number of distribution functions per cell
n	Particle density
N_f	Full number of distribution functions
p	Macroscopic pressure
Q	Collision integral
T	Temperature

Performance Model Symbols

B_c	Code balance
B_m	Machine balance
B_x	Spatial block size in x direction
B_y	Spatial block size in y direction
$B_{m,core}$	Single core machine balance
T_x	Duration of an x loop update
T_y	Duration of a y loop update
T_z	Duration of a z loop update
T_{1Cload}	Duration of transfers from L1 cache to core

$T_{1Cstore}$	Duration of transfers from core to L1 cache
T_{21}	Duration of L2 to L1 cache transfers
T_{32}	Duration of L3 to L2 cache transfers
$T_{collect}$	Duration of a collection loop update
T_{core}	Duration of arithmetic operations
T_{IACA}	Durations determined through IACA
T_{M3}	Duration of memory to L3 cache transfers
T_{total}	Total duration of transfers and operations
T_{x-peel}	Duration of an x-peel loop update
$T_{x-remainder}$	Duration of an x-remainder loop update

1 Introduction

1.1 Motivation

Computational fluid dynamics (CFD) is an essential tool in engineering that has significantly altered many fields of application. The steady increase in capabilities of hard- and software and ongoing development of simulation methods has made CFD an irreplaceable tool and which will continue to gain in importance. It enables the creation of highly optimized design, speeds up and reduces cost of development and fosters understanding of processes that cannot be examined in experiments.

Starting from the beginnings of numerical solution of greatly simplified flow problems in the 1930s, the growth in computational complexity of the simulation models has kept pace with the growth in transistor counts described by the well-known Moore's law. In fact the use of computational power can be increased almost arbitrarily through finer resolution, detailed modeling or more complex interactions. In this context a fully resolved direct numerical simulation (DNS) of an airplane in high lift configuration has been estimated as plausible in 2080 [139]. However, a more recent publication, that takes into account the stagnation of Moore's law, estimates it as much later or even never [140], showcasing the need for fast fluid simulation techniques.

Fast CFD methods are essential in achieving such major goals. But apart from these "Grand Challenges" there is a large demand for fast fluid simulations in almost every field of engineering. The number of applications for flow simulations are countless, covering fields as diverse as aerospace [7,35], weather forecast [143], sports [26,51], plasma physics [136], microfluidics [43], bio-medical [100, 127], and even the film and game industry [108, 149]. The importance of fast fluid solvers is further emphasized when hundreds of simulation runs are required, as may be the case when CFD is embedded in advanced frameworks, such as uncertainty quantification [10] or design optimization [110].

A popular subclass of CFD are the incompressible flows, which this work focuses on. Furthermore all investigations are performed with regards to engineering application, for which often only trends are required rather than exact predictions. Frequently only qualitative flow patterns are meaningful as simulations and experiments are unlikely to match up exactly due to many sources of uncertainty that cannot be captured by the idealized computational environments [99]. For these cases a fast solution in terms of runtime with an accuracy limited to few significant digits is more important than high convergence orders or errors at the level of machine precision. It is exactly that point in which the present work differs from many performance investigations published in literature. They are usually carried out on a new method or an improvement of one to demonstrate their competitiveness. Typically these works present high convergence orders or update rates for an application that caters to the strengths of the presented method. This work strives to look at a broader and more practical sense of performance by considering the total time to solution at several coarser levels of accuracy and independent of promoting a particular class of methods.

There are many covert ways in which some performance demonstrations fall short of practical applicability and which this work seeks to avoid. The performance figures that are given as a by product of other findings are in particular danger of some of these shortcomings. Occasionally these are carried out as comparisons to competing approaches. Apart from a certain bias towards the own method that may arise the example can be tuned to better suit the presented technique in subtle ways or more effort went into optimizing one of the approaches. By designing the test cases to be compatible with any involved approach and extensive tuning of all codes this effect can be minimized. This is often the case with publications dedicated to performance comparisons, however the findings can lack practical applicability in other ways. A commonly used metric for comparisons that provides a very striking figure are the updates per second. It also avoids the very difficult task of quantifying the result quality, but it is exactly that which is necessary to show performance. After all it is the result per time that defines absolute performance in practice. In this work all comparisons are geared towards the total time required to achieve identical solution qualities. All design decisions that go into the method choice and setup as well as test cases and performance measures are carried out on this basis. The results are performance figures that can be compared even between fundamentally different methods. Furthermore these results are in a form that can be applied directly to questions relevant to engineering as for example the required time to a particular accuracy or alternatively the achievable accuracy within a fixed time frame. All performance comparisons carried out in this work are performed under this philosophy of relevance to the engineering practice.

Performance comparisons over the entire field of incompressible flows are naturally not feasible as the number of configurations goes into the millions and determining the absolute highest performance would require testing all possible combinations against one another. Even subsets, e.g. for a particular problem setting or limited to specific methods are still far beyond what can be handled computationally today. Much of this work is therefore concerned with reducing the number of combinations that have to be tested while maintaining a maximum of significance and without compromising the fairness. The required choices fall into several categories. Restrictions applied a priori greatly reduce the number of parameters as the study can be focused on a particular branch of CFD, reducing the scope of this work. All these types of decisions come at the cost of generality. A further category are the decisions derived from known behavior and trends. Experienced users know the strength and weaknesses of different methods and can select techniques based on the case at hand, e.g. the suitability of implicit or explicit solvers can often be judged reasonably well, given the case is sufficiently far from the break even point. Of course it is this middle ground, which offers the most interesting insights. Hence such apparent decision and the corresponding biased test cases should be avoided as much as possible. However, the choices can be augmented by small-scale testing, which allows for informed choices even near the balance point. An unerring way to determine configuration parameters is through full scale testing. It ensures the highest performance within the scope set forth by the previous limitations. As this is computationally expensive and the effort grows exponentially with the number of parameters, only a very limited number of choices can be treated in this manner.

The exact scope of this work is outlined in the following section, but to motivate further topics treated in this work some of these restrictions are introduced here. They are categorized in three major domains that maximize their independence and which is a concept this entire work follows. These domains are the algorithmic and numerical methods, the hardware and according

code optimization, as well as the test cases. This distinction allows for concise treatments of the performance aspects within each of these units. Interactions are referenced but the full treatment of the often iterative solutions to this interdependence is not shown. However, for true high performance all three categories must match up exactly as for example the method must be the most suitable for a particular test case and be suitable for tuning on a specific hardware platform.

Defining the scope of the test case selection determines the types of flows for the high performance claim is valid. It is restricted to incompressible non-turbulent single phase flows at relatively low Reynolds numbers, i.e. laminar and unsteady flows for simple geometries. This class of flows is often encountered in engineering, but is a narrow enough field to be covered by a single method. The group of hardware and code optimization are straight forward. The hardware is given as a very powerful server platform that allows for a wide range of problems to be simulated and the tuning is performed according to that architecture. All measurements are carried out on the same machine to maximize reproducibility.

Continuing with the selection of simulation techniques it is clear that every method to be tested requires considerable work to implement and optimize, not to mention familiarization with its characteristics. Practicality dictates that an in depth comparison has to be limited to two methods, but even then the range of options that can be explored is immense. The lattice Boltzmann method and the artificial compressibility finite difference method selected for this work come from fundamentally different backgrounds, but exhibit a very similar behavior. This aids in creating a fair comparison and allows for additional investigations apart from the high performance aspect.

This particular choice of methods in motivated further in the following. The lattice Boltzmann method (LBM) is considered to be a very fast approach by many and has demonstrated its suitability to many fields of applications [4, 31, 87, 95]. Compared to traditional solvers of the Navier-Stokes equations (NSE) it is a relatively recent development [34, 55, 85, 104, 123], yet it has matured enough for commercial packages to be available [32]. It is derived from a gas kinetic approach which through coarse graining has been adapted for solving macroscopic flows. This combination of high performance and a fundamentally different approach makes it an excellent candidate for a comparison to a classical flow solver. The additional question whether there are general performance differences between these approaches is also covered by this work. Specifically if it is the underlying approach that makes LBM implementations fast or if that can be attributed to other factors such as the restriction to a minimal feature set and consequent tuning. Or in other words, can classical approaches based on discretizing the NSE be constructed to reach comparable performance. There are two major factors that indicate direct solvers to the NSE should be competitive. The LBM approaches the incompressible Navier-Stokes equations in the asymptotic limit [75, 76, 164] and quadratic convergence can only be proven for very specific flow configurations without practical relevance [77]. A further indicator lies in the design of modern hardware that tends to be bandwidth (BW) limited for computational codes due to the bandwidth gap [101]. As the LBM utilizes 19 stored quantities per cell, compared to the four common in traditional approaches, it is likely to cause bandwidth saturation at a lower throughput.

The counterpart to the LBM is a finite difference method (FDM) as an explicit artificial compressibility scheme. Its convergence behavior and accuracy have been found to be very similar [37, 52, 102] and FD stencils have even been derived from lattice Boltzmann collision operators directly [73]. This is in line with the previously stated goals of using similar methods

to determine the break even point and the true origins of supposedly high performance. It also reduces the complexities in the comparison setup and facilitates fair measurements.

1.2 Scope and objectives

The wide number of interesting questions and challenges which have been touched upon in Section 1.1 are refined in the following to give the exact scope of this work. The overall topic is of course the pursuit of the fastest possible incompressible flow solvers, with respect to practical engineering requirements. In this context performance is determined by the total time to solution for a set quality of solution rather than convergence orders and update rates. This topic is often neglected when showing performance as establishing fair and method independent measures and the thorough optimization of all the involved implementations is outside reasonable efforts for performance figures shown alongside a published method. Establishing and realizing the many steps required for a fair comparison is a major objective of this work. This includes the selection of suitable methods for the comparison as testing all potential methods against each other is out of the question.

Some of the restrictions placed on this investigation from the outset is the analysis of single phase incompressible flow in a laminar or unsteady state. Effects such as turbulence or low Reynolds numbers as for microfluidics are excluded. Within this regime two methods have been chosen for a direct comparison, namely the lattice Boltzmann and artificial compressibility finite difference method. Among other criteria these have been selected as likely candidates for very high performance. By means of these two approaches the specific and general steps towards a fair comparison are determined. A truly unbiased and universal result could only be guaranteed if every possible combination of hardware, method, implementation and test case were tested. However, this is infeasible even for small portions of the CFD field and the search space must therefore be reduced. This is achieved by a series of design decisions based on experience, selective testing and reduction of the scope for which the comparison is significant. As the effort put into these decisions here is much greater than it would be even in performance oriented development, the compared performance is reasonably higher than it would be for practical application. Additionally the performance gains become marginal after a certain level of optimization which leads to very small deviations in any case.

The LBM and artificial compressibility FDM were not only chosen for their performance, but also for their similarities. This aspect is not only important towards a fair comparison, but it also allows a second question to be addressed in this work. Is the presumably high performance of the LBM really a matter of the fundamentally different approach compared to the FDM or is it a matter of configuration and implementation eased by the restricted feature set? In other words, could the artificial compressibility method (ACM) with its similar characteristics be tuned to a comparable performance level by applying analog restrictions and implementation strategies? For either of these two main topics the same steps towards an unbiased comparison are needed. For compactness these are subdivided into the three categories of the methods, the code tuning and the testing. An overview of what these categories encompass is given in the following.

The methods have been determined already, however surrounding these basic approaches there are essentially unlimited number of choices leading to their full definition. The process of selecting the algorithmic details has to balance high performance and retaining the similarity

in their behavior. The former is crucial to distinguishing the impact of the LBM itself from its implementation. Additionally, the results on the middle ground for evenly matched methods offer more insights than merely confirming well known performance properties.

The next topic is that of the implementation and hardware specific tuning and is treated comprehensively. It is relatively distinct from the other topics and mostly straightforward. Given the method and hardware it is an exercise of successively applying optimizations until maximum performance is achieved. Nonetheless it is essential as a well tuned implementation can gain orders of magnitude in performance and undetected inefficiencies can skew the outcome entirely, giving methods an unjustified bad reputation. In order to avoid such effects the code tuning is investigated thoroughly and augmented by a variety of performance models. As noted earlier all timing comparisons are carried out on a single platform. However within the scope of this work some transferability to other hardware architectures is explored to give an indication on the wider picture of fast CFD.

An even larger impact on the performance evaluation has the test case choice. Not only does it define the field of application for which the fastest method is searched, but it can also have subtle and hidden influences on the performance. Arguably a test case can be constructed to show superior performance for any method. However, demonstrating either methods predominance is not as insightful as establishing the performance boundary between approaches. In the present work this region is investigated through a series of increasingly complex test cases. These tests are derived and the methods tuned correspondingly by using a combination of a priori decisions, theoretical considerations, selective testing and the extensive automated full scale optimization introduced above. Furthermore, not only the final performance figures are shown, but they are also placed in the overall fast CFD context by shedding some light on the performance trends as well. All this is carried out with regard to engineering practices, reinforcing the relevance to practical application.

The findings from the test case measurements are used answer the questions posed above and to place the performance of the LBM and FDM into relation with each other and into the wider picture of fast CFD. Additionally secondary factors that are not directly performance related but are still important to engineering applications such as ease of use are discussed. In combination an assessment of the suitable fields of application for both methods is carried out.

In summary the measures take in this work go to great lengths to achieve impartial absolute performance figures and this sets it apart from other published performance analyses. Every decision and step taken is geared towards this goal. First of all the performance is measured as a quantifiable result quality per runtime. Then the methods used for the comparison are explicitly selected for their speed and comparability. Both implementations are optimized extensively and specific to the employed hardware. Further, the test cases are selected to minimize the bias and to provide quantifiable errors. Finally the simulation parameters of either implementation that are not bound by the test case setup are optimized exhaustively for every combination of test and target error. This allows for the fairest comparison of the real performance of the selected LBM and FDM in absolute numbers at a level previously unknown. This work further addresses some of the soft factors that go into the simulation method selection and which is not commonly found in literature.

1.3 Overview

This work is structured according to the three categories (method, code, testing) discussed earlier. In Chapter 2 the basics to the finite difference and lattice Boltzmann solvers are introduced. It highlights the origins of each method and provides the basic equations that are solved. This shows the fundamentally different approaches that solve the same incompressible flow.

Chapter 3 expands upon the methods and shows the derivation of the numerical approaches from their mathematical background. First the lattice Boltzmann method which discretizes the Boltzmann equation derived in the previous chapter is treated. The artificial compressibility finite difference method is then introduced with a focus on the analogies to the LBM. This chapter also gives implementation details that are essential to high performance and form the basis of the tuning in following chapter.

Chapter 4 is dedicated to performance optimization for both implementations. To this end a look at the relevant hardware features is given and three performance models are introduced. They allow the assessment of the achieved performance gains at varying degrees of accuracy, which is also subject of investigation here. Then a series of optimization steps are applied and their impact on performance is assessed. With the aid of the performance models the final results of the tuning process are investigated. The remaining bottlenecks are addressed by comprehensive code parameter optimization.

In Chapter 5 the actual comparison is carried out. With the foundation of carefully chosen and configured methods and extensive tuning the test case selection process is given. The reasons for particular choices are discussed and the exact specifications per test case are defined. This includes flow parameters, comparison criteria and target tolerances. For each case both methods are analyzed and tuned individually and finally the results are set into relation to each other. The suit of test cases consists of three steady-state flows, namely a duct flow (5.3.1), a lid driven cavity (LDC) (5.3.2.1) and a laminar flow around a cylinder with a square cross section (5.3.3). These are followed up by unsteady (5.4.1) and transient (5.4.2) flows around a cylinder with a square cross section which compares the methods in a more suitable flow configuration for explicit solvers. Finally the results from all test cases are evaluated and both methods are set into relation to each other, not only in terms of performance, but also further factors relevant to practical uses.

The findings of this work are summarized in Chapter 6 and an outlook to the most promising extensions to this work is given.

2 Fundamentals of incompressible flows

In reality there are no truly incompressible flows. Every material is compressible on some scale and the speed of sound in every medium is finite. However, it can be an excellent approximation in many situations and eliminates the necessity to track very fast acoustic propagation. Incompressible flows correspond to an infinite speed of sound, which causes the pressure and velocity fields to adapt instantaneously and which in turn permits the use of different solvers. Whether it is appropriate to assume incompressibility is therefore as much a matter of the flow properties as the materials. Typically flows up to $\mathrm{Ma} = {}^{U_{\max}}/_{c_s} \leq 0.3$ are accepted as incompressible, where $U_{\max}$ is the maximum velocity in the entire domain and c_s the speed of sound in the medium. Additionally, a constant density is assumed. Thermal effects can lead to a variable density even in incompressible settings, which are simulated using the Boussinesq approximation. However, this work is limited to isothermal flows of Newtonian fluids, hence all densities and viscosities are considered constant. The methods that are established later are solvers for incompressible flows, however they reintroduce some form of compressibility to alter the solving properties.

In the following, two fundamentally different approaches to the mathematical description of incompressible flows are given. The traditional technique lies in setting up partial differential equations (PDE), which describe the fluid at a macroscopic level. With the appropriate initial and boundary conditions these can be solved numerically. For a few particular setups analytical solutions are known as well. While this approach can be considered a top-down approach, there is also the option of a bottom-up approach. For this the molecules of which a fluid is composed at microscopic level are considered directly. The number of particles and interactions on that level are too large for most practical applications. Hence, this view is coarse grained through statistical mechanics until a system is obtained, which allows macroscopic flows for engineering purposes to be simulated.

2.1 Navier-Stokes equations

The traditional approach to simulating incompressible flows is by numerically solving the incompressible Navier-Stokes equations. These are a set of well-known PDEs, which describe the behavior of fluid flows on a macroscopic level. They were originally derived by [29, 109, 122, 144], but for a detailed introduction and derivation the reader is referred to [6, 154]. The NSE are then discretized spatially and temporally and solved for a given set of boundary and initial conditions. There are many variations on how this is realized for countless academic and commercial fluid simulation tools. As the derivation of the equations has been exercised by many publications, only a short overview shall be given here. The introduction starts out with the compressible NSE, which serves as a reference for the artificial compressibility method introduced in Section 3.2.

The NSE are based on the concept of conservation of mass, momentum, and energy. In the following these are derived for an arbitrary fluid volume V. However, as the focus lies on isothermal flows, the energy conservation is not considered.

Conservation of momentum is based on the assumption that temporal changes in momentum for said volume can only result from body forces $\boldsymbol{f}_b$ acting on the volume V or surface forces $\boldsymbol{f}_s$ acting on volume V's surface S

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \rho\boldsymbol{u}\,\mathrm{d}V = \int_V \boldsymbol{f}_b\,\mathrm{d}V + \int_S \boldsymbol{f}_s\,\mathrm{d}S. \tag{2.1}$$

Rearranging with the transport and divergence theorem, the equation can be written as

$$\int_V \left(\frac{\partial\rho\boldsymbol{u}}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\otimes\boldsymbol{u})\right)\mathrm{d}V = \int_V (\rho\boldsymbol{f}_b + \nabla\cdot\boldsymbol{\tau})\,\mathrm{d}V, \tag{2.2}$$

where $\boldsymbol{\tau}$ is the stress tensor representing the surface forces on the V through the relation $\boldsymbol{f}_s = \boldsymbol{\tau}\boldsymbol{n}$ with the surface normals $\boldsymbol{n}$. As equation (2.2) holds for every choice of V, it can be expressed in its differential version

$$\frac{\partial\rho\boldsymbol{u}}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\otimes\boldsymbol{u}) = \rho\boldsymbol{f}_b + \nabla\cdot\boldsymbol{\tau} \tag{2.3}$$

as well. In order to complete the system of equations for the case of Newtonian fluids, the constitutive relation, following [6],

$$\boldsymbol{\tau} = -p\mathbf{I} + 2\mu\left(\boldsymbol{\epsilon}(\boldsymbol{u}) - \frac{1}{3}(\nabla\cdot\boldsymbol{u})\,\mathbf{I}\right) \tag{2.4}$$

is used. It incorporates the rate of strain tensor $\boldsymbol{\epsilon}(\boldsymbol{u}) = \frac{1}{2}\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right)$. As mentioned in Section 2, only a constant dynamic viscosity μ is considered in this work. This allows for the following description of the conservation of momentum for compressible, isothermal, Newtonian fluids

$$\frac{\partial\rho\boldsymbol{u}}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\otimes\boldsymbol{u}) = -\nabla p + 2\mu\nabla\cdot\left(\boldsymbol{\epsilon}(\boldsymbol{u}) - \frac{1}{3}(\nabla\cdot\boldsymbol{u})\,\mathbf{I}\right) + \rho\boldsymbol{f}_b. \tag{2.5}$$

Conservation of mass is the second component to the Navier-Stokes equations. The equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \rho\,\mathrm{d}V = \int_V s\,\mathrm{d}V \tag{2.6}$$

governs the rate of change in mass to be equal to the mass added through sources s. For the plain single phase flows considered here no source terms are present, i.e. $s = 0$. As equation (2.6) holds for arbitrary volumes V and through the application of the transport theorem the differential form

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}) = 0 \tag{2.7}$$

of the continuity equation for compressible flows is obtained. In conjunction with equation (2.5), they form one configuration of the compressible Navier-Stokes equations.

Incompressible flows allow for a simplification of the Navier-Stokes equations. The general assumptions leading to incompressible flow were described in Section 2. The consequences of fluid velocities much lower than the speed of sound and when there are no external density altering effects (e.g. temperature changes) the total derivative of the density vanishes $\frac{\mathrm{D}\rho}{\mathrm{D}t} = 0$. This allows the Navier-Stokes equations to be rewritten as

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

$$\nabla\cdot\boldsymbol{u} = 0. \tag{2.9}$$

A difficulty, which arises with the incompressible NSE, is the change from a hyperbolic-parabolic type equation to an elliptic-parabolic type [82]. A method for circumventing this issue is given in the Section 3.2 on the numerical treatment.

Under consideration of the incompressibility constraints the viscous term can be further modified to the convective form

$$\rho\frac{\partial \boldsymbol{u}}{\partial t} + \rho\boldsymbol{u}\cdot\nabla\boldsymbol{u} + \nabla p - \mu\Delta\boldsymbol{u} = \rho\boldsymbol{f}_b, \tag{2.10}$$

where Δ represents the Laplace operator. This form is the basis for the discretization carried out later. Special treatments for stability and convergence are method specific and are introduced alongside the numerical solver.

The Navier-Stokes equations can be transformed into a dimensionless form, which then depends on few dimensionless parameters and decouples the solution of the NSE from the physical problem. Given geometric similarity, the Reynolds number

$$\mathrm{Re} = \frac{UL}{\nu} \tag{2.11}$$

with L the characteristic length scale, U the characteristic velocity and $\nu = \frac{\mu}{\rho}$ the kinematic viscosity, is the most meaningful dimensionless quantity for incompressible flows. It correlates inertial to viscous forces and is used to characterize flow patterns. In case of compressible flows, including the weakly compressible methods introduced later, the Mach number is also relevant, as it describes the ratio of maximum flow velocity and the speed of sound. Simulations that are independent of physical dimensions seem very attractive for a comparison between fundamentally different approaches. However, due to the different ways in which the simulations are nondimensionalized this could impact the results, in particular through the different round-off errors the methods exhibit at different magnitudes. As almost all simulations in engineering are tied to an actual physical problem, a comparison with the full dimensions is appropriate. Nonetheless it is important to use these dimensionless numbers to ensure that either method solves the same problem.

Boundary condition choice is essential to the well-posedness of the problem. Here, the two most basic boundary conditions are used, namely Dirichlet and Neumann conditions. These are usually straightforward to apply for discretized Navier-Stokes equations as the Dirichlet conditions prescribe the velocity values directly and Neumann conditions the tractions. For the explicit time integration scheme that is used later, it is advantageous to prescribe the pressure instead, which also takes care of the constant pressure mode. It gives rise to tricky nomenclature as the Neumann boundary condition at the level of the NSE is actually a Dirichlet condition at the level of the pressure field [80, 117].

2.2 Statistical mechanics

The alternative to discretizing the PDEs outlined above is the bottom-up approach starting at a molecular level of the fluid. This enables a completely different set of methods to be applied to fluid simulations. At its core there is the interaction of millions of molecules (or atoms), which constantly attract and repulse each other in oscillatory movement, leading to a macroscopic flow. Naturally, this description is very accurate, as it takes into account the physics at a very detailed level, just short of quantum mechanical effects. The attractive and repulsive forces are the result of electrostatic and van der Waals forces acting between molecules and which lead to complex and costly N-body interactions [44, 148]. While this approach is very accurate and inherently incorporates many fluid dynamic effects, it is also very noisy in regards to the macroscopic values, which are therefore obtained by averaging thousands of particles. This renders direct simulation not feasible for macroscopic flows. In fact 1 m^3 of air at 0 °C contains $2.69 \cdot 10^{25}$ molecules [164], which is beyond current supercomputing capabilities and which drastically increases further for liquids.

However, for certain purposes molecular dynamics (MD) simulations are used very successfully, though at much smaller space and time scales than required in this work. It is also a good starting point for coarse graining, which allows for much larger systems to be simulated at the expense of detail. As the macroscopic behavior is of interest, approximations of the underlying particle behavior are perfectly reasonable and allow for drastic simulation cost savings. The lattice gas cellular automata (LGCA) are given as an example and to introduce the concept and foundation for further considerations. They are the predecessors to the lattice-Boltzmann method, which is the eventual goal of this summary. The LGCA represents particles as either 0 or 1 and which travel a distance of 1 on a lattice in every time step. On the lattice cells collision rules are evaluated for all incoming particles that are then sent out in a potentially different direction. This is a very coarse representation of the processes used in MD, but it enables the use of orders of magnitude more particles. At the same time, these large numbers of particles are required for the averaging process through which the macroscopic values are obtained. Even with very large numbers of particles these values remain noisy, which is the reason that LGCA are not typically used nowadays. The lattice-Boltzmann method is an advancement of the LGCA retaining similar lattice based concepts. The particles of a boolean type are replaced by continuous densities, which eliminates the noise in the primary quantities. For the LBM the motivation for the numerical method and the mathematics supporting it are more closely intertwined than the discretization and derivation for traditional NSE based approaches. Nonetheless, the mathematical considerations are presented here, separated from the implementation details in Section 3.1 as far as possible.

The following is only a brief summary of the origin of the Boltzmann equation with Bhatnagar, Gross, Krook (BGK) approximation. It can be found in much more detail in [19, 20, 145, 164]. The starting point is the realization that accounting for all interaction between every particle is not feasible. They are instead replaced by a limited number N_f of distribution functions. These functions are denoted as $f_{N_f}\left(\boldsymbol{q}_1, \boldsymbol{p}_1, \ldots, \boldsymbol{q}_{N_f}, \boldsymbol{p}_{N_f}, t\right)$, where $\boldsymbol{q}_i$ and $\boldsymbol{p}_i$ are coordinates and momentum of particle i. The evolution of these distribution functions over time are governed by the Liouville equation, which is essentially a continuity equation for the distribution function in phase space [131]. The Boltzmann equation is obtained by approximating the Liouville equation,

resulting in

$$\frac{\partial f}{\partial t} + \boldsymbol{v}\frac{\partial f}{\partial \boldsymbol{x}} + \frac{\boldsymbol{K}}{m}\frac{\partial f}{\partial \boldsymbol{v}} = Q(f), \tag{2.12}$$

where $f(\boldsymbol{x}, \boldsymbol{v}, t)$ is the distribution function for a single particle, with $\boldsymbol{x} = \boldsymbol{q}_i$, $\boldsymbol{v} = \boldsymbol{p}_i/m$, particle mass m and a collision integral $Q(f)$. For simplicity, the external forces $\boldsymbol{K}$ are not considered further. Several assumptions allow the collision integral to be formulated more explicitly. These assumptions are, that only two particles collide at any time, the velocities before the collision were independent and local collision is not influenced by external forces. This is known as the molecular chaos assumption. The collision integral for the particles a and b can then be written as

$$Q(f_a, f_b) = \iint \sigma(\Omega)\,|\boldsymbol{v}_a - \boldsymbol{v}_b|\,\left[f(\boldsymbol{v}'_a)f(\boldsymbol{v}'_b) - f(\boldsymbol{v}_a)f(\boldsymbol{v}_b)\right]\,\mathrm{d}\Omega\,\mathrm{d}^3\boldsymbol{v}_a\,\mathrm{d}^3\boldsymbol{v}_b. \tag{2.13}$$

It links outgoing velocities $\boldsymbol{v}'$ with incoming ones $\boldsymbol{v}$, using the differential collision cross section $\sigma(\Omega)$. As this integral is difficult to handle, an approximation $J(f)$ was devised, known as the BGK model [9, 153]

$$J(f) = \omega\left[f^M(\boldsymbol{x}, \boldsymbol{v}, t) - f(\boldsymbol{x}, \boldsymbol{v}, t)\right]. \tag{2.14}$$

It is also referred to as a single relaxation time (SRT) model, since it relaxes the distribution functions towards the local Maxwellian $f^M(\boldsymbol{x}, \boldsymbol{v}, t)$ for every collision. The local Maxwellian is in turn derived from the Maxwell distribution

$$f^{(M)}(\boldsymbol{x}, \boldsymbol{v}, t) = f(\boldsymbol{x}, \boldsymbol{v}, t) = n\left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \exp\left[\frac{-m}{2k_B T}(\boldsymbol{v} - \boldsymbol{u})^2\right], \tag{2.15}$$

which is a special solution to the collision integral. The distribution depends on the temperature $T(\boldsymbol{x})$, the density $n(\boldsymbol{x})$ and the mean velocity $\boldsymbol{u}(\boldsymbol{x}) = \frac{1}{n}\int \boldsymbol{v} f(\boldsymbol{x}, \boldsymbol{v}, t)\,\mathrm{d}^3 v$. All further steps in (numerically) solving these equations and how the macroscopic flow values are recovered from these density distributions are given in Section 3.1. For the proof that the incompressible Navier-Stokes equations are recovered in the asymptotic limit refer to [75] or for the traditional analysis through the Chapman-Enskog expansion towards the (weakly compressible) Navier-Stokes equations to [34].

Boundary and initial conditions differ significantly from their NSE solver counterparts. Typically, these conditions are applied in terms of macroscopic values, which is generally difficult in statistical mechanics, as it can be challenge on its own to find a microscopic ensemble which accurately represents these macroscopic specifications. Furthermore, these ensembles are not unique due to the large number of unknowns that have to be determined from few prescribed values. This requires additional constraints for the behavior on the microscopic level and for which, at least in the LBM context, no definite set of rules has been found yet. There are some types of boundary conditions to which these difficulties do not apply, as periodic domains and symmetry conditions are trivial to realize on a particle level by displacing or mirroring the set of particles crossing the boundary.

3 Numerical approaches

As determined in the previous chapter, the problem of simulating incompressible fluid flows can be approached from two directions. These are either through discretizing PDEs that describe the fluid behavior, i.e. the Navier-Stokes equations, or by coarse-graining the low level interactions of fluid particles. While these approaches seem very different, some numerical methods with similar behavior from both areas can be found in the intermediate regime. In the following, one of these methods from either side is presented. This encompasses the considerations on the simulation techniques at a high level and their position within the topic of fast CFD, as well as particular implementation choices.

The lattice-Boltzmann method is introduced first, since it is more restricted in the method options. The cause for the smaller number of options is that using methods based on kinetic theory to simulate macroscopic problems is still relatively new. The NSE based top-down approaches have been in use for much longer and are still the main focus of research. There are countless techniques to choose from, whether it is implicit, semi-implicit, or explicit methods or down to detailed stabilization parameter determination. This gives the flexibility to pick an approach that best matches the characteristics of the LBM and which is essential for a detailed comparison. The remaining choices are still extensive and are selected to form a fast method.

3.1 Lattice Boltzmann method

As far as CFD methods originating from gas-kinetic descriptions are concerned, there are few options when simulating flows common in engineering. The direct simulation via molecular dynamics or other methods on the microscopic level are orders of magnitude too computationally expensive. Even simulations on the mesoscopic scale, such as dissipative particle dynamics or direct simulation Monte Carlo, are not capable of simulating practical tasks like a flow around a vehicle. For these dimensions further coarse-graining is required and the most established approach in that regard is the lattice Boltzmann method.

Though the LBM is a rather recent development [145], it is an established CFD method and is being actively researched by many groups. It is a versatile foundation for different kinds of applications, e.g. turbulent, multi-phase, porous or suspension flows [2, 22, 118]. Apart from these diverse uses, it is also in the focus of research on high performance computing [121, 159, 160]. As described in Section 2.2, the LBM is derived in a fundamentally different fashion from traditional CFD methods based on discretizing the Navier-Stokes equations. It is therefore interesting to investigate whether this might be the reason for the supposedly excellent performance. At the same time, it is a simplified gas-kinetic description, which could induce behavior that deviates from the expected solution of the incompressible Navier-Stokes equations. After all, the NSE are only approached in the asymptotic limit [62, 74] and convergence cannot always be ensured [75].

An introduction to the origin of the LBM has already been given in Section 2.2. That approach is continued here to the full, discretized form. Additionally, some indications on the simulation behavior and an outlook to the implementation choices are given.

The lattice Boltzmann method is derived from the Boltzmann equation

$$\frac{\partial f}{\partial t} + \boldsymbol{v}\nabla f = Q\left(f, f^{(eq)}\right), \tag{3.1}$$

introduced earlier (2.12), with external forces omitted for clarity. The density distribution functions f and the velocity $\boldsymbol{v}$ are still continuous and Q is a collision integral for which a suitable choice is inserted later, along with the equilibrium distribution function $f^{(eq)}$. Discretizing the distribution functions f_i and the velocity $\boldsymbol{v}_i$ by direction i yields the discretized Boltzmann equation

$$\frac{\partial f_i}{\partial t} + \boldsymbol{v}_i\nabla f_i = Q\left(f_i, f_i^{(eq)}\right). \tag{3.2}$$

The number of distribution functions or directions N are determined by the lattice that is introduced later. When space and time are also confined to a lattice, the cell spacing $\Delta x = {}^1/_{\delta_x}$ and time step size $\Delta t = {}^1/_{\delta_t}$ result in a lattice velocity $c = \frac{\Delta x}{\Delta t}$ and the derivatives in equation (3.2) can be approximated by differences. Conversion to a dimensionless form with characteristic length L, velocity U, density n_r and time between collisions t_c and inserting the lattice velocity $\hat{\boldsymbol{c}}_i = \frac{\boldsymbol{v}_i}{U}$, gradient $\hat{\nabla} = L\nabla$, time $\hat{t} = t\cdot\frac{U}{L}$, and density distribution function $\hat{f}_i = \frac{f_i}{n_r}$, the lattice Boltzmann equation

$$\hat{f}_i(\hat{\boldsymbol{x}} + \hat{\boldsymbol{c}}_i\Delta\hat{t}, \hat{t} + \Delta\hat{t}) - \hat{f}_i(\hat{\boldsymbol{x}}, \hat{t}) = \hat{Q}\left(\hat{f}_i(\hat{\boldsymbol{x}}, \hat{t}), \hat{f}^{(eq)}(\rho, \boldsymbol{u})\right) \tag{3.3}$$

emerges. For legibility the $(\hat{\cdot})$ is omitted from here on. The relationship between the discrete density distribution functions f_i and the macroscopic quantities of interest are

$$\rho(\boldsymbol{x}, t) = \sum_{i=0}^{N-1} f_i(\boldsymbol{x}, t), \qquad \boldsymbol{j}(\boldsymbol{x}, t) = \rho(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=1}^{N-1} f_i(\boldsymbol{x}, t)\boldsymbol{c}_i. \tag{3.4}$$

The pressure p is linked directly to the density ρ due to the weakly compressible behavior and is derived later.

A closer look at equation (3.3) shows that a time step can naturally be split into a collision and a streaming part

$$\tilde{f}_i(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) + Q\left(f_i(\boldsymbol{x}, t), f^{(eq)}(\rho, \boldsymbol{u})\right) \tag{3.5}$$

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = \tilde{f}_i(\boldsymbol{x}, t). \tag{3.6}$$

The distribution function in direction i is modified by the collision operator, calculated from all current incoming distribution functions, and is then streamed into the new position $\boldsymbol{x} + \boldsymbol{c}_i\Delta t$ at the new time step.

The choice of the lattice is very important for the LBM. It needs to fulfill several basic requirements like symmetries and allow for Galilei-invariance. This gave rise to many three dimensional lattices, the most popular being D3Q15, D3Q19 and D3Q27. The number following

the D denotes the number of space dimensions and the number following the Q specifies the number of distribution functions per cell N. The case of D3Q19 is depicted in Figure 3.1. It is a common choice and offers a balance with higher accuracy than D3Q15 at lower computational effort and memory occupation than D3Q27 or higher. This makes it a suitable compromise between throughput and accuracy required for the comparison. The $N = 19$ density distribution functions are oriented and named

$$\{E, W\} = (\pm 1, 0, 0)^T, \ \{N, S\} = (0, \pm 1, 0)^T, \ \{T, B\} = (0, 0, \pm 1)^T \tag{3.7}$$

and ordered as the lattice velocities

$$\begin{aligned} \boldsymbol{c}_0 &= (0, 0, 0)^T \\ \boldsymbol{c}_i &= \{NE, N, NW, W, SW, S, SE, E, T, TE, \\ &\quad TN, TW, TS, B, BE, BN, BW, BS\}, \qquad i = 1, \ldots, 18. \end{aligned}$$

The speed of sound c_s is a lattice constant and takes on $c_s^2 = \frac{1}{3}$ for the D3Q19 lattice.

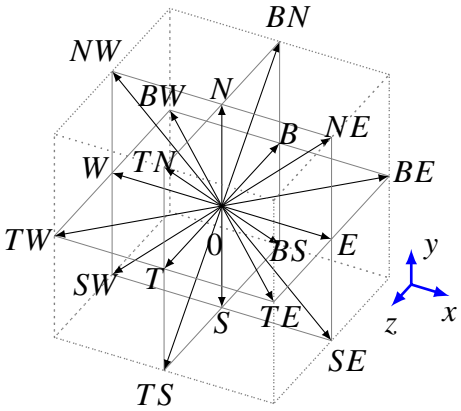


Figure 3.1 Density distribution function directions for a D3Q19 lattice.

In LBM the physics is governed by the choice of the collision operator. It is also responsible for most of the computational complexity. The most basic operator is the lattice Bhatnagar-Gross-Krook (LBGK) [123] operator, which is a linear single-relaxation-time model. Though it has some deficiencies, it is very useful for demonstrating the derivation of the collision operator and it is later expanded upon to counteract some of its shortcomings.

The LBGK approximation to the collision operator is

$$Q \approx J\left(f_i, f_i^{(eq)}\right) = \lambda\left(f_i - f_i^{(eq)}(\rho, \boldsymbol{u})\right), \tag{3.8}$$

with the relaxation time λ and the local equilibrium distribution functions $f_i^{(eq)}$. The equilibrium distribution functions are the product of Maxwell distributions and symmetry considerations of the velocity moments. It takes the form of

$$f_i^{(eq)}(\rho, \boldsymbol{u}) = \begin{cases} \rho\left(1 - 2c_s^2\right) - 5\rho_0 \|\boldsymbol{u}\|^2, & \text{for } i = 0 \\ w_i\left(\rho c_s^2 + \rho_0\left(\boldsymbol{u} \cdot \boldsymbol{c}_i\right) + \frac{3}{2}\rho_0\left(\boldsymbol{u} \cdot \boldsymbol{c}_i\right)^2 - \frac{1}{2}\rho_0 \|\boldsymbol{u}\|^2\right), & \text{for } i = 1, \ldots, N \end{cases} \tag{3.9}$$

with a constant density ρ_0 treated later and a lattice and direction specific weight w_i. The weights

$$w_i = \begin{cases} \frac{1}{6}, & \text{if } \|\boldsymbol{c}_i\| = 1 \\ \frac{1}{12}, & \text{else} \end{cases} \tag{3.10}$$

are obtained through the Chapman-Enskog expansion [39, 42], which allows the relationship between the LBM on a specific lattice and the Navier-Stokes equations to be determined. It also provides an expression for the pressure

$$p = c_s^2 \rho. \tag{3.11}$$

The pressure $p = p_0 + \delta p$ and density $\rho = \rho_0 + \delta\rho$ are separated into a constant and deviatoric part. As the base pressure and density level can be selected through the reference parameters, ρ_0 is chosen as 1, which eliminates it from many equations, reducing the computational complexity and improving numerical round off errors.

While the introduced components of the lattice Boltzmann equation and the linear SRT approximation operator for the collision operator already form a full lattice Boltzmann method, apart from boundary conditions, it suffers from significant drawbacks. The simplicity of the approach makes it attractive to high performance optimizations, but the impact on the simulated physics are not negligible, as for example the viscosity depends on the mesh size. Moreover, the popular mid-way bounce-back boundary condition is not guaranteed to be centered in between cells [40], and a slow convergence to steady state solutions has been observed [118]. This can be circumvented by the use of multi-relaxation-time (MRT) approaches. These offer a separate relaxation time for every particle population and thereby great control over the collision operation. A drawback is the separate evaluation of collision terms according to the distinct relaxation parameters and which prevents the elision of many terms, resulting in a performance penalty. A middle-ground is the two-relaxation-time (TRT) approach [41]. The use of two parameters allows for most terms to be merged, while solving the issues of the SRT method.

In the following, the SRT collision operation introduced earlier is expanded to a TRT type. At the core of the TRT model is the split of the collision operator into a symmetric (+) and an anti-symmetric (−) part. The respective relaxation times are λ_e and λ_o, which recover the SRT method when chosen equal. The equations governing the TRT collision are

$$Q_i = \lambda_e \left(f_i^+ - e_i^+\right) + \lambda_o \left(f_i^- - e_i^-\right), \tag{3.12}$$

$$f_i^\pm = \frac{1}{2}\left(f_i \pm f_{\bar{\imath}}\right), \tag{3.13}$$

$$e_i^+ = w_i \left(c_s^2 \rho + \frac{1}{2}\left(3(\boldsymbol{u}\cdot\boldsymbol{c}_i)^2 - \|\boldsymbol{u}\|^2\right)\right), \tag{3.14}$$

$$e_i^- = w_i \left(\boldsymbol{u}\cdot\boldsymbol{c}_i\right), \tag{3.15}$$

$$e_0^\pm = \rho - \sum_{i=1}^{N-1} e_i^+. \tag{3.16}$$

The direction $\bar{\imath}$ denotes the opposite of i, i.e., $\boldsymbol{c}_i = -\boldsymbol{c}_{\bar{\imath}}$ and w_i are the same lattice specific weights as introduced in equation (3.10) for the D3Q19 lattice.

Inserting the operator (3.12) into the lattice Boltzmann equation (3.3), the full system of equations is obtained

$$\tilde{f}_i(\boldsymbol{x}, t+1) = f_i(\boldsymbol{x} - \boldsymbol{c}_i, t) \tag{3.17}$$

$$f_i(\boldsymbol{x}, t) = \tilde{f}_i(\boldsymbol{x}, t) + \lambda_e p_i + \lambda_o m_i, \tag{3.18}$$

with

$$p_i = \frac{1}{2}\left[f_i + f_{\bar{i}} - w_i\left(2c_s^2\rho + 3\left(\boldsymbol{u}\cdot\boldsymbol{c}_i\right)^2 - \|\boldsymbol{u}\|^2\right)\right] \tag{3.19}$$

and

$$m_i = \frac{1}{2}\left[f_i - f_{\bar{i}} - w_i 2\left(\boldsymbol{u}\cdot\boldsymbol{c}_i\right)\right]. \tag{3.20}$$

The use of the intermediate distribution function $\tilde{f}$ allows the split of the evolution equation into two steps, the "streaming" in equation (3.17) and the "collision" in equation (3.18). The implications of this split are discussed alongside the implementation considerations. Application of the Chapman-Enskog expansion to the TRT model additionally yields the relationship between relaxation time and viscosity ν,

$$\lambda_e = \frac{-2}{6\nu + 1}. \tag{3.21}$$

Only the symmetric part of the relaxation parameter λ_e is tied to the viscosity. The anti-symmetric relaxation parameter λ_o is a free parameter which is governed by criteria outside a physical interpretation. A beneficial choice is presented alongside the boundary conditions.

Up to this point external body-forces have been neglected. This was mostly for brevity, but also because incorporation of body forces is not unique for the lattice Boltzmann method [15, 45, 106, 133]. The first of the three techniques is through incorporating the (globally acting) external force in the pressure gradient, which results in a modified term for the density variation in the equilibrium distribution. A similar approach is the modification of the momentum that is supplied to the equilibrium distribution. Thirdly, the distribution functions can be modified post-collision by a separate term. All approaches have different limitations and a combination is required to reproduce the NSE for non-zero velocities and non-zero nonlinear terms [15], though it has been suggested this might not be enough [45]. As the body-force is very invasive to integrate into the LBM and the possibility of deviations from the NSE results exists, they are not used in the comparison examples and not treated any further.

The resulting TRT scheme is a fast CFD simulation technique that approaches the incompressible Navier-Stokes equations in the asymptotic limit [75, 76]. The behavior is of a weakly compressible nature, which comes with special characteristics that have to be taken into account. Most significantly, there can be artificial acoustic modes and the small compressibility can have a significant impact on the results. The ratio between the flow velocity and the artificial speed of sound is governed by the artificial Mach number and has to be carefully selected to find the optimal balance between accuracy and throughput. This characteristic greatly complicates a fair comparison. The finite difference technique used for the comparison is selected for a similarity in this behavior, which alleviates this difficulty.

The LBM boundary conditions have to be applied at a gas-kinetic level as well. This means that traditional Dirichlet and Neumann boundary conditions cannot be applied directly as for NSE based simulations. This lead to the development of a large number of boundary conditions,

some of the most popular are given in [41, 94]. In the following, the general concept behind boundary conditions in LBM is introduced, along with the specific types of conditions used in the later examples.

Most boundary conditions originate from the so called bounce-back (BB) scheme. The simplest approach is a mid-way bounce-back condition, which simply returns outgoing density distributions in the opposite direction in the following time step $f_{\bar{i}}(\boldsymbol{x}, t+1) = f_i(\boldsymbol{x}, t)$. As the name implies this results in a boundary condition that resides in between the outer and first layer of cells and acts as a no-slip condition. Unfortunately, there are some drawbacks to this technique. Even though the mass the seem to be conserved by returning the outgoing densities, this is not guaranteed at a global scale [24]. This technique, in its plain form, can also result in a small velocity error, which can either be interpreted as an offset from the exact middle of a link or as a residual slip velocity on the boundary conditions [53, 60, 94]. The exact error depends on the velocity at the boundary, which makes it difficult to handle. There are techniques which apply an opposing slip velocity in order to reduce the error [60, 61, 172], but they add significant overhead to the otherwise very simple boundary condition, which makes them ill suited for this performance comparison. Fortunately, through the use of the two-relaxation-time technique, there is a different option to achieve accurate no-slip condition. The second relaxation time λ_o for the collision operator was left open to a suitable choice. By using the "magic" parameter $\Lambda_{eo} = {}^3/_{16}$ [41] to determine the anti-symmetric component

$$\lambda_o = \frac{2\lambda_e + 4}{(4\Lambda_{eo} - 1)\,\lambda_e - 2}, \tag{3.22}$$

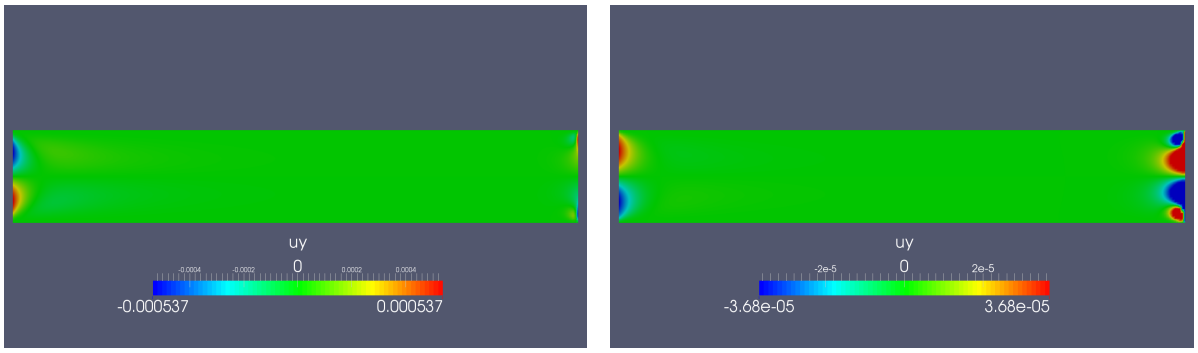
an exactly centered BB scheme is ensured.

Boundary conditions other than no-slip conditions are even more problematic to impose. The main difficulty lies in the 19 distribution functions in use here, but only three of the four primary values can be prescribed. Even considering certain symmetries that must be fulfilled by the density distributions, the system is under constrained. There are various approaches to creating additional constraints, which then typically lead to over constrained systems. These are in turn fulfilled in a weak sense. Naturally, many different techniques have been developed which offer many choices on the best compromise between computational effort and accuracy.

The in- and outflow condition presented here tend to be basic, as more complicated schemes are at risk to hinder tuning for highest performance. For the inflow sides a standard bounce-back is applied first. This leads to a no-slip state, i.e., $\boldsymbol{u} = \boldsymbol{0}$. Then the incoming velocity component $\boldsymbol{u}_{bc}$, evaluated at the mid-way boundary crossing position $\boldsymbol{x}_{bc} = \boldsymbol{x} + \frac{1}{2}\boldsymbol{c}_i$ and the current local density are inserted into the equilibrium distribution. The resulting incoming density distribution functions

$$\tilde{f}_i(\boldsymbol{x}, t) = f_{\bar{i}}(\boldsymbol{x}, t) + w_i \rho \left(\boldsymbol{u}_{bc}\big(\boldsymbol{x} + \frac{1}{2}\boldsymbol{c}_i, t+1\big) \cdot \boldsymbol{c}_i \right) \quad \forall\, i \text{ intersecting the BC.} \tag{3.23}$$

replace the previously undetermined ones and can be transported in the streaming step like any regular distribution function. As mentioned before, there are Dirichlet conditions that provide higher order accuracy or are able to better deal with the anisotropic effects introduced by non-constant inflow velocity profiles. A lightweight approach in this regard are the correction terms introduced in [54]. Even though they do not completely eliminate the cross-flows induced behind a parabolic inflow profile, the magnitude was significantly reduced as shown in Figure 3.2.



(a) Without correction terms **(b)** With correction terms [54]

Figure 3.2 Cross velocities induced behind the parabolic inflow for a plane poiseuille flow. The outflow is unchanged and demonstrates the differences in scaling.

For the outflow boundaries a related approach is used, where instead of one of the three velocities the pressure is prescribed. The relation from equation (3.11) is used to convert the pressure into density variation, which are prescribed instead. The chosen approach is named pressure anti-bounce-back (PAB) in [42]. The undetermined incoming distribution functions are again constructed from the reflected outgoing distribution function $f_{\bar{i}}(\boldsymbol{x}+\boldsymbol{c}_{\bar{i}}, t)$ and the equilibrium state of the symmetric component from equation (3.14). The missing velocities required for $e_i^+(\rho, \boldsymbol{u})$ are obtained by extrapolation from the inner boundary cell. The undetermined incoming distribution function $\tilde{f}_i$ can then be described through

$$\tilde{f}_i(\boldsymbol{x}, t) = -f_{\bar{i}}(\boldsymbol{x} + \boldsymbol{c}_{\bar{i}}, t) + w_i \left(c_s^2 \rho + \frac{1}{2} \left(3(\boldsymbol{u} \cdot \boldsymbol{c}_i)^2 - \|\boldsymbol{u}\|^2 \right) \right). \tag{3.24}$$

The recovery of the incompressible Navier-Stokes equations through the lattice Boltzmann method in the asymptotic limit is shown with the Chapman-Enskog expansion [34, 42] or by asymptotic analysis [75, 76]. Due to its weakly compressible nature the LBM is not inherently time-accurate. A common remedy in classical NSE solvers are dual-time-stepping schemes, but these can only be applied to complex hybrid lattice Boltzmann methods with significantly more computational effort [48]. A great advantage of the LBM is its wide stability range despite its explicit nature. Stability has been proven for a wide range of flow velocities [164], but the rate of convergence has not. Research has shown that second order convergence in velocity and pressure for linear collision operators can only be guaranteed if special requirements are met [77]. Specifically, the solution must be smooth, on fully periodic domains, and the initial field must be either quiet or at least of fourth order in the distribution functions in relation to the velocity, which is in practice computationally very costly. If any of the aforementioned conditions is violated, the order can be reduced. For bounded domains, which in practice is almost always the case, the convergence is generally reduced. Especially in conjunction with the ubiquitous bounce-back boundary condition the convergence order is reduced to first order accuracy in the velocity and the pressure is generally inconsistent [76]. Another problematic issue are the initial conditions. In engineering practice the general flow pattern is usually known and convergence can be accelerated by setting an approximate initial field or even restarting from a previous simulation. Unfortunately, this is difficult to realize for the LBM, as the initial field must be of high order

accuracy. Such an initial field is very expensive to compute and therefore all comparisons are started from a quiet field.

The LBM code used in this work is based on a highly tuned implementation developed by the International Lattice Boltzmann Development Consortium (ILBDC) [170]. It implements the two-relaxation-time scheme and uses the "magic" parameter for bounce-back accuracy. The stream and collision operation are arranged in a pull order, i.e. the stream and subsequent collision are merged into one loop. This reduces the number of memory transfers, which are a common bottleneck on modern hardware platforms. The density functions in the fluid cells are stored in a 1-dimensional array in arbitrary order. Access to the appropriate density functions of the own fluid cell (for collision) and the neighboring fluid cells (for propagation) are performed through a separate list of connectivity information [171]. With this approach it is straight forward to merge the collision and propagation into a single loop over the field. Another advantage of the indirect addressing is, that in the context of complex geometries the lattice cells that are outside the computational domain can be skipped altogether.

The ILBDC code is parallelized with OpenMP in a ccNUMA aware fashion. The performance on caching hardware architectures is further increased through a strip-mining approach to the loop kernel evaluation routine. Additionally, the density distribution arrays are toggled between time steps, which eliminates spatial data dependencies and, in conjunction with the pull-scheme, read-for-ownership (RFO) can be eliminated.

3.2 Finite difference artificial compressibility method

The second component to the comparison in this work is a flow solver using the traditional approach of solving the discretized incompressible Navier-Stokes equations. There are countless possibilities in the techniques that can be used. In order to be able to compare both approaches more directly methods with a similar behavior to the LBM are required. This aids further in distinguishing the performance impact of the underlying lattice Boltzmann approach from the implementation and test case choice. Use of a similar technique has the added benefit of being receptive to comparable optimization techniques.

A fully explicit solver for the weakly compressible Navier-Stokes equations [25, 90, 125, 126, 141] is used. This artificial compressibility finite difference method is interchangeably referred to as either ACM or FDM in the following. The incompressible NSE are altered by addition of a partial derivative of the pressure to the continuity equation (2.9)

$$\frac{1}{c_s^2}\frac{\partial p}{\partial t} + \rho \nabla \cdot \boldsymbol{u} = 0. \tag{3.25}$$

The derivative is scaled with the artificial speed of sound c_s and the divergence term is multiplied with the density ρ for consistency in the units. This addition to the continuity equation transforms the elliptic-parabolic type system back into hyperbolic-parabolic type equations, which allows the use of solvers for the compressible Navier-Stokes equations [82]. In case of steady-state flows the modified NSE still recover the fully incompressible state as the temporal derivative vanishes. For unsteady and transient flows this is not the case and an isothermal flow is solved instead [52]. However, the impact of the compressibility can be user controlled by adjusting the artificial speed of sound c_s and thus setting the artificial Mach number Ma. A dual-time-stepping scheme could

be used to eliminate the influence of the compressibility as a time-steady subproblem is solved for every physical time step [105]. Since the LBM offers no equivalent technique it is not used here. Neither is the application of the ACM to turbulent flows [98], which is outside the scope of this work.

The artificial compressibility method exhibits a close resemblance to the LB method [52]. It has been used in comparisons with the LBM and its predecessor LGCA before [8, 37, 52, 102], but not with rigorous regard to the total time to solution, i.e. the practically usable performance. One of the reasons for the high performance of the ACM is its fully explicit nature. No linear system has to be solved implicitly and costly sparse matrix vector products can be avoided. They are usually very memory bandwidth intensive, which in turn is often a bottleneck in simulation codes for PDEs [101, 120]. This reason also gives rise to the consideration that the FDM could potentially be faster than the LBM since only four unknowns per point as opposed to the 19 unknowns must be transferred across the limited bandwidth interface.

The weakly compressible Navier-Stokes equations are discretized in convective form. Separating the time derivative and the residuals $\boldsymbol{R}$ and R the equations become

$$\rho \frac{\partial \boldsymbol{u}}{\partial t} = \underbrace{-\rho \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nabla p + \mu \Delta \boldsymbol{u} + D_u(\boldsymbol{u}) + \rho \boldsymbol{f}}_{\boldsymbol{R}(\boldsymbol{u},p,t)} \tag{3.26}$$

$$\frac{1}{c_s^2} \frac{\partial p}{\partial t} = \underbrace{-\rho \nabla \cdot u + D_p(p)}_{R(\boldsymbol{u},p,t)}. \tag{3.27}$$

with the density ρ, kinematic viscosity μ and body force $\boldsymbol{f}$. The terms D_u and D_p are stabilization terms, which are provided later.

An explicit Runge-Kutta (RK) scheme is applied for the temporal discretization. In order to reduce the memory consumption and the load on the memory bandwidth, a low-storage or 2N variant based on [17, 162] is used:

$$U^{t+1} = U^t + \Delta U_s^{t+1}, \tag{3.28}$$

$$\Delta U_i^{t+1} = B_i \Delta t F(U^t + \Delta U_{i-1}^{t+1}), \quad i = 1, 2, \ldots, s \tag{3.29}$$

$$\Delta U_0^{t+1} = 0. \tag{3.30}$$

For the momentum equation U are the unknowns $\boldsymbol{u}$, F is the residual $\boldsymbol{R}$ and B_i the factor α_i/ρ. Analog for the continuity equation U is the pressure p, F the residual R and the factor B_i is given as $\alpha_i c_s^2$. The coefficients α_i are $\alpha_i = \frac{1}{s+1-i}$. This approach makes use of an arbitrary number of RK stages trivial, but it shows a slightly reduced increase in convergence orders per stage, compared to full RK schemes. It was shown in [162] that 2N-storage, fourth-order schemes require at least five stages, except for integrating special functions. Also the third-order 2N-storage schemes show reduced accuracy [17]. As an alternative 3N-storage schemes do not exhibit these limitations [36], but are not considered due to the increased storage requirements. In this work a two stage, second-order version is used, as preliminary testing showed that higher orders are unlikely to pay off for the coarse tolerances targeted here.

The choice of an explicit scheme requires that the CFL criterion must be met. Hence the time step size Δt is limited to

$$\Delta t = \frac{CFL \cdot \Delta x}{(c_s + \max(|\boldsymbol{u}|))}. \tag{3.31}$$

The quantity Δx denotes the mesh size of the difference stencil given below.

The time-discrete equations are discretized in space through finite differences on a globally uniform rectangular grid, i.e. $\Delta x = \Delta y = \Delta z$. This matches the discretization possibilities of the selected LBM. An additional benefit is the reduced number of floating point operations (FLOP) per stencil evaluation, as the mesh size can be factored out. Second order central differences are used in this work, as they offer a good memory bandwidth to FLOP ratio. In Chapter 4 it is shown, that with all extensions and optimizations included it matches the bandwidth to arithmetic performance ratio of the test platform. The terms that approximate equations (3.26)-(3.27) are described separately, according to

$$\boldsymbol{R}(\boldsymbol{u}, p, t) = \boldsymbol{R}_{adv}(\boldsymbol{u}, t) + \boldsymbol{R}_{presgrad}(p, t) + \boldsymbol{R}_{laplace}(\boldsymbol{u}, t) + \boldsymbol{R}_{momentstab}(\boldsymbol{u}, t), \tag{3.32}$$

$$R(\boldsymbol{u}, p, t) = R_{div}(\boldsymbol{u}, t) + R_{contistab}(p, t) \tag{3.33}$$

in the following, though the code uses a merged and optimized version of the loop kernel. The terms in the momentum equations are given exemplary for the velocity component u_x. The pressure gradient and the Laplace term are

$$R_{u_x,presgrad} = \frac{-1}{2\Delta x}\left(p^{i+1jk} - p^{i-1jk}\right) \tag{3.34}$$

$$R_{u_x,laplace} = \frac{\mu}{\Delta x^2}\left(u_x^{i+1jk} + u_x^{i-1jk} + u_x^{ij+1k} + u_x^{ij-1k} + u_x^{ijk+1} + u_x^{ijk-1} - 6u_x^{ijk}\right). \tag{3.35}$$

The advection term requires additional treatment, since the velocity u_x on the edge between points $\boldsymbol{x}^{ijk}$ and $\boldsymbol{x}^{i+1jk}$ should be identical, independent of which side the advection term is evaluated on. This is achieved by writing the term as

$$\begin{aligned} R_{u_x,adv} = \frac{-\rho}{4\Delta x}\Big[&\left(u_x^{i+1jk} + u_x^{ijk}\right)\left(u_x^{i+1jk} + u_x^{ijk}\right) - \left(u_x^{ijk} + u_x^{i-1jk}\right)\left(u_x^{ijk} + u_x^{i-1jk}\right) \\ &+ \left(u_x^{ij+1k} + u_x^{ijk}\right)\left(u_y^{i+1jk} + u_y^{ijk}\right) - \left(u_x^{ijk} + u_x^{ij-1k}\right)\left(u_y^{ijk} + u_y^{i-1jk}\right) \\ &+ \left(u_x^{ijk+1} + u_x^{ijk}\right)\left(u_z^{i+1jk} + u_z^{ijk}\right) - \left(u_x^{ijk} + u_x^{ijk-1}\right)\left(u_z^{ijk} + u_z^{i-1jk}\right)\Big]. \end{aligned} \tag{3.36}$$

The above is a simple scheme that is very compact with a stencil width of only two. A drawback of the scheme is that it requires stabilization, which is implemented as an adaption of [71] using only the fourth order artificial viscosity (AV) term since there are no shocks in incompressible flows. The modified term is given as

$$R_{u_x,momentstab} = c_{vel}\frac{\rho \mathrm{Re}_{cell}}{2\Delta x}|u_x^{ijk}|\left(-u_x^{i+2jk} + 4u_x^{i+1jk} - 6u_x^{ijk} + 4u_x^{i-1jk} - u_x^{i-2jk}\right) \tag{3.37}$$

with $\mathrm{Re}_{cell} = \max(0, \min(1, \frac{|u_x^{ijk}|\Delta x}{\nu} - 1))$. The parameter c_{vel} governs the influence of the stabilization and is user configurable. The exact choice depends on the particular simulation and is part of the simulation parameter optimization performed for the test cases 5.3.1 - 5.4.2. Many alternative stabilization techniques have been developed over the decades, but an investigation into their performance characteristics is outside the scope of this work. Also, AV has been a popular choice for high performance computations in a variety of different realizations in the past [25, 59, 71, 93, 97, 166]. The major drawback of the fourth order AV is the increased stencil width, which incurs additional memory transfer costs and complicates efficient cache usage.

The continuity equation is treated analog to the momentum equations. The discretized divergence term is

$$R_{p,div} = \frac{-\rho}{2\Delta x}\left[(u_x^{i+1jk} - u_x^{i-1jk}) + (u_y^{ij+1k} - u_y^{ij-1k}) + (u_z^{ijk+1} - u_z^{ijk-1})\right] \tag{3.38}$$

and the fourth order AV is

$$\begin{aligned} R_{p,contistab} = \frac{c_{pres}}{2\Delta x(c_s + |u^{ijk}|)}\big(& -p^{i+2jk} + 4p^{i+1jk} - 6p^{ijk} + 4p^{i-1jk} - p^{i-2jk} \\ & - p^{ij+2k} + 4p^{ij+1k} - 6p^{ijk} + 4p^{ij-1k} - p^{ij-2k} \\ & - p^{ijk+2} + 4p^{ijk+1} - 6p^{ijk} + 4p^{ijk-1} - p^{ijk-2}\big), \end{aligned} \tag{3.39}$$

where c_{pres} is again a free parameter.

The Dirichlet and Neumann boundary conditions are straight forward compared to the many possible versions for the LBM. As the velocity and pressure are directly operated on and the NSE are advanced in time explicitly it is trivial to prescribe the desired values on the boundary points. It should be noted that a Neumann type boundary condition in the NSE context is actually realized as a Dirichlet condition on the pressure within the context of the continuity equation. Here boundary conditions are prescribed with the help of halo points that pad the actual simulation domain. Figure 3.3 shows a series of points perpendicular to the boundary on which, depending on the type of boundary condition here exemplary for a prescribed velocity, the value on the boundary (0) is prescribed, the first halo point (−1) is extrapolated maintaining a constant first derivative, the second halo point (−2) using the second derivative, and so on. Alternatively gradients may be prescribed directly as for symmetry conditions and it can also be applied to the pressure field. The use of halo points allows the points on the boundary to be evaluated using the

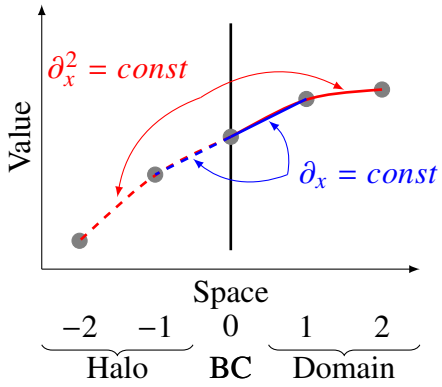


Figure 3.3 Extrapolation onto halo points from the inner points of the domain for integrated treatment of boundary conditions. The number of halo points depends on the stencil width.

regular stencil, albeit at a lower order. The order reduction is necessary for stability, however globally the full convergence order can potentially still be achieved [46,47]. Furthermore it allows the use of a single long loop over the entire domain, which reduces overhead for parallelization via OpenMP or for offloading to GPUs [96] and simplifies the tuning techniques introduced in Chapter 4.

The aforementioned scheme is implemented in C++ with the time critical section, requiring fine grained control over memory access and vectorization in C. The code “Fastflo” is adapted from the implementation used in [96], but with additional performance enhancements and purely focused on highest throughput for the tasks presented later. The memory layout is of Array of Structures of Arrays (AoSoA) type with a block size fixed at compile time. This allows for easy adaptation to different hardware, while retaining efficient vectorization and good data locality for improved cache usage. Additionally this memory layout avoids over-straining the translation lookaside buffer (TLB), which can easily occur for large stencils as used here and can severely limit performance. The AoSoA memory layout is incorporated in a spatial blocking scheme for optimized caching and the full stencil presented above is implemented as a semi-stencil. This splits the stencil into multiple parts and is also known as coordinate splitting (CS). It can significantly reduce register pressure. An in depth analysis of these techniques is given in Chapter 4, which also gives detailed performance figures that show the competitiveness with regards to the LBM.

4 Performance optimization

In this chapter the code tuning process is shown and analyzed for the FDM and LBM code. A subset has also been published in [157] and some excerpts are used in the following. The tuning plays an essential role in the performance comparison. The possible speedups from a straight forward implementation can be orders of magnitude and therefore alter the outcome of the comparison altogether. The performance optimization is largely independent from the other steps towards fast CFD as it is mostly dictated by the chosen method and the testing platform. Naturally the method itself must be selected suitable to the hardware and should offer the potential to efficient programming, but the low-level optimization presented in the following is mostly reactive.

Here, the full performance optimization process is shown in detail. First an overview over current hardware structures is given with a focus on the aspects important to the optimization. This is followed up by another prerequisite, which are the performance models. These are used to asses the potential speedup and to verify the optimization steps recover the expected performance. They also offer insights into the cause of unexpected performance breakdowns. Generally, they are used in an iterative process with the different tuning steps. For the sake of brevity, the performance modeling and the tuning steps are presented separately. Next the optimization process itself is presented, broken down into separate steps, some of which only show their benefits in combination or are only applicable to either the LBM or FDM. It should be noted that the chosen steps are pursued only to within a reasonable limit of the absolute performance optimum. There are several reasons for this, apart from the diminishing returns at the high end. It retains some flexibility in porting the code to other platforms when all optimizations are restricted to guiding the compiler to more efficient code, rather than applying hand coded assembly. The missed gains are marginal, especially since modern compilers can produce near optimal code when guided. The gradual buildup of techniques demonstrates the interaction with particular hardware features. In order to place the performance optimization within the bigger picture of CFD the most relevant steps are additionally tried on different platforms. The first is similar to, but older than the reference platform and showcases the development in hardware architecture over the last years. The other is a much different Intel Many Integrated Core (MIC) platform for which only some of the used techniques bring benefits. The fully optimized versions of the FDM and LBM are verified with a sequence of performance models. These increase in complexity and accuracy and thereby reduce the error between estimated and measured performance. Nonetheless the fully optimized codes show significant discrepancies even for the most advanced model. In order to determine the cause of the accuracy breakdown the execution-cache-memory (ECM) model is applied to the FDM and analyzed in detail. This helps identifying modeling effort to insight gains ratio and the practical boundaries of performance estimates. The scope of the tuning process not only covers the use of certain techniques, but also the ideal adjustment of the accompanying parameters such as loop blocking sizes. The range in which these can be configured and the number of parameters lead to very large search spaces for

the best configuration. The performance models are used to determine a suitable configuration based on theoretical considerations. However, due to the limited accuracy of the models it is followed up by a full search for the ideal configuration to ensure maximum performance for the comparison.

It should be noted that the explicit nature of the compared simulation methods makes the update rates for both the LBM and FDM independent of the specific simulation setup and results, apart from the domain dimensions. This allows optimal configurations to be tabulated per domain dimensions and hardware configuration. The platform that is used for performance optimization and the later time measurements of the test cases is an Intel Xeon E5-2690 v4 dual socket system. The main features of the cores and cache hierarchy are summarized in Table 4.1. The features relevant to the tuning process are also described in more detail in section 4.1.1.

Compute	
Number of cores	14
Base frequency	2600 MHz
Vectorization	4x (AVX2)
Execution units	8
floating point operation	1 cyc
fused multiply add	0.5 cyc
vectorized divide	16 cyc
load	2×32 B/cyc
store	32 B/cyc

(a) Core

L1 cache	
size	32 KiB per core
cacheline size	64 B
associativity	8 way
BW to L2 cache	64 B/cyc half-duplex
L2 cache	
size	256 KiB per core
cacheline size	64 B
associativity	8 way
inclusive	no
BW to L3 cache	32 B/cyc half-duplex
L3 cache	
size	35 MiB shared
cacheline size	64 B
associativity	20 way
inclusive	yes
BW to MM	76.8 GB/s

(b) Cache hierarchy

Table 4.1 Selected features of the Intel Xeon E5-2690 v4 CPU [65, 68] and the corresponding theoretical performance figures. It is used in a dual-socket configuration for the optimization and performance measurement. Some important aspects of the design are not disclosed by the hardware manufacturer, resulting in contradicting information. Additionally, the effective performance may be considerably lower [58].

The system has a total of 512 GiB DDR4 main memory (MM) at 2400 MHz which is evenly split between the four memory channels of each socket. Both sockets combined provide a theoretical peak bandwidth of 153.6 GB/s. The system is configured to run in cluster-on-die (COD) mode with turbo boost disabled and the CPU is forced to constantly run at full speed (performance governor) and not to go into a sleep state (forced C0 state). Additionally the uncore clock frequency is configured to maintain the highest speed through BIOS configuration. The constant clock speeds are verified by monitoring the number of executed cycles with the builtin

performance counters. All of the afore mentioned configurations aim to reduce noise in the measurements and helps performance modeling due to the constant clock speeds and lack of core speed interdependence.

4.1 Hardware and performance aspects

4.1.1 Hardware architecture introduction

This section gives a general introduction to the hardware and performance aspects in high performance computing. The most important hardware features in this regard and their interactions are presented. This creates the basis for the optimization techniques introduced in section 4.3. Furthermore, the concept of cache and memory bandwidth and arithmetic throughput are established. The balance between these two per hardware and code form the basis for the performance modeling introduced in Section 4.2. All examples and numbers given in the following are derived from the reference platform used for the comparison. It is a dual socket platform for which all hardware details important to the later analysis are provided in Table 4.1. Though the properties given in the following are specific to a certain CPU, the concepts can generally be transferred to other general purpose CPUs. Many can even be applied to different types of architectures, such as the many-in-core or the advanced RISC machine (ARM) platforms. The optimization techniques presented later can often be ported over as well, but as the performance characteristics change these have to be adjusted individually.

Up to this day most general purpose computers are setup as the combination of the compute core and shared data and instruction memory. This goes back to the von Neumann architecture which was first been conceived in 1945 [150]. This combination gives rise to two main types of performance bottlenecks. Compute bound, when execution is limited by the cores ability to execute instructions or memory bound, when the memory cannot supply instruction and data to the core fast enough, resulting in a starvation and wasted stall cycles of the core. From that very basic starting point many enhancements have been developed through the decades to overcome not only these two limitations. However, even the systems in use today, which have become extremely complex beyond the apprehension of a single person and with transistor counts exceeding 1.4 billion even for the older Intel Ivy-Bridge architecture [63], they still adhere to the same basic concept. Historically the single core systems were enhanced by continuous increases in the clock frequency. Eventually physical limits where reached with this approach, as signal propagation speed, power requirement and thermal management issues arose [103]. Such systems are not competitive when energy consumption plays a major role, either for ecological or mobility reasons. Development has therefore turned towards parallel and more complex systems to yield the desired performance improvements [12]. As not all components of CPUs are equally suited to this treatment and a general difficulty directly relevant to the performance considerations in this work has developed. The memory transfer rates have not kept up with the performance increase of the cores, which lead to the so called bandwidth gap [101] that still persists in systems today.

To bridge this performance gap caches are used. These are placed close to the core and scale with the number of cores. They allow for very fast access to data and instruction as they run at the same or similar clock frequencies as the cores themselves. Generally, they are small and use volatile memory types, which allow for very high speeds. On the reference platform each

L1 cache has a capacity of 32 KiB and a bandwidth of approximately 166 GB/s, which in total over all cores is 30 times faster than the main memory. This is shown in Figure 4.1 in term of the bandwidth and latencies in relation to the distance from the core. The cache is generally

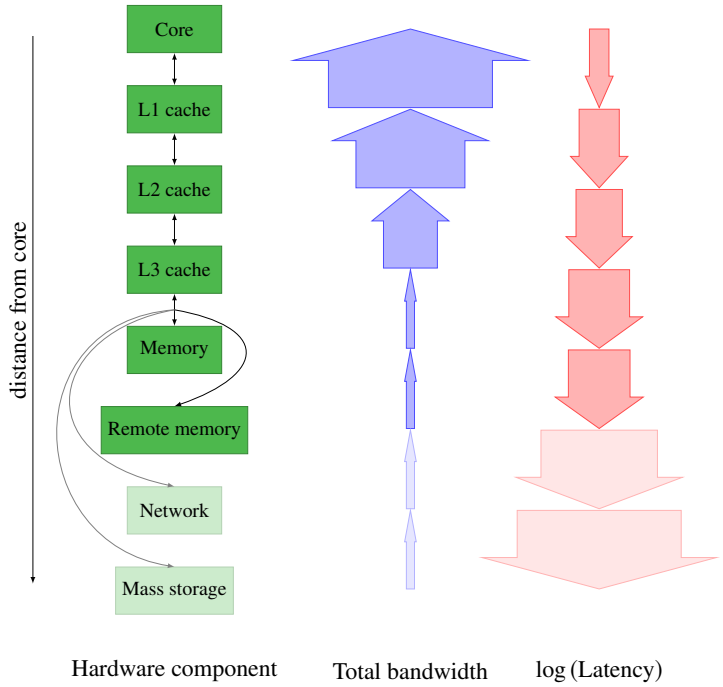


Figure 4.1 Bandwidth and latency over the logical distance from the core for components in the memory hierarchy. The width of the arrows corresponds to the bandwidth over all cores and the logarithm of the latency of the reference platform. The opaque parts are not to scale.

transparent to the user and acts without user intervention. However, the cache efficiency can be greatly influenced through particular programming techniques. Caches are organized in cache lines (CL), which on the reference platform is 64 B wide. Whenever the core tries to access the memory, whether for reading or writing, this request is instead passed to the cache which will try to serve the request instead. If that particular cache line is not available in the cache already, it is loaded from memory. The cache line resides in the cache until the space is required for a different memory region, which causes the cache line to be evicted. This means it is written back to memory if the content was altered or simply discarded otherwise. This is known as a write-back policy and is only one of many caching strategies [135]. A great advantage to this technique is that multiple accesses to the same address can be served from the cache directly, freeing up main memory bandwidth, and due to the broad cache line size the adjacent storage will typically be available already. The k-way cache used on the reference platform is a very common choice as it allows several cache properties to be balanced. This design has some characteristics that should be considered when programming or tuning for performance. For a better understanding a small example on how the cache is used is given. In a one-way cache the decision into which cache line the data is loaded is based on the modulo of the physical memory address. If a cache can hold eight cache lines, every 512th address maps to the same cache line. This leaves little room for reuse of cache data, as the generally random memory address have a high chance of replacing a cache line that will be used again soon. The k-way strategies allows for this to be balanced

by providing k cache lines for every address modulo. This way a separate strategy can be used to decide which of the k cache lines is replaced when a conflicting address is loaded. A typical choice is the one that has not been accessed the longest.

With the help of caching the effective bandwidth available to the core can be increased greatly, while at the same time reducing the load on the main memory interface. To fully utilize the greater bandwidth the computational throughput must be increased accordingly. One way to achieve this is by employing single instruction multiple data (SIMD), also known as vectorization. This allows a single instruction to be applied to multiple values at once, usually in the same number of cycles as for the scalar operation. In theory the throughput scales linearly in the vectorization width. Hence the width has been increased steadily and a width of 512 B (or 8 double, 16 floats, or 32 integers) is being established. The reference platform uses a width of 4 doubles, which is associated with the advanced vector extensions (AVX(2)) instruction set. The theoretical linear scaling is in practice impeded by several prerequisites for efficient vectorization. The most important is the memory alignment. The data to which the SIMD instructions are applied have to be in consecutive order, otherwise they need to be rearranged beforehand by other instructions, which incurs overhead in CPU cycles and cache lines. Ideally the address of the first data element is aligned to 64 B, enabling the faster aligned loads and stores. These requirements have to be considered when programming as current compiler have limited capabilities of automatically inferring memory alignment and the performance gains from vectorization may be lost. The drastically increased computational throughput that can be achieved however, must be balanced by even higher bandwidth. One approach is the creation of a hierarchy of caches as it is depicted in Figure 4.1. In general, the cache size and performance are linked in a reciprocal fashion. This means that the fast caches are usually small and are placed close to the core, while slower but larger caches are positioned further up in the hierarchy. This allows for a compromise of large amounts of cache, which promotes reuse, and fast caches to fulfill the core's bandwidth requirement. These caches can be exclusive or inclusive, which means that a particular cache line is held only in one cache level or in all down to the current one simultaneous, respectively. Also the write strategy, such as write back, described earlier, or write-through, for which every change is immediately propagated through all levels, can lead to complex interactions. These configurations are potentially different for every platform and must be considered when optimizing. The specific cache configuration for the reference platform is given in Table 4.1.

One major drawback of a large and complex cache hierarchy is that every level increases the access latency to the main memory. As the request for data and the data itself must generally be passed from level to level the latency increases with every step as shown in Figure 4.1. As a counter measure many techniques have been developed to hide these latencies. On most platforms the cores can detect repeated access pattern to the memory and will request the expected memory regions to be prefetched to the cache. The occasional misspredict ion causes some overhead, but it can be kept to a minimum through programming for regular and compact access patterns. Not all memory accesses can be ordered in such a fashion, hence additional techniques are used. Pipelining for instance breaks down complex instructions into smaller instructions that can be handled faster and simultaneously, e.g. fetching data may overlap with the addition of the previous instruction and which in turn overlaps with the store of the instruction before that. Places the predecoded instructions in instruction queues allows for the data requirements to be analyzed before the instruction is actually processed. Typically only few instructions can be processed

ahead of time due to dependencies on earlier instructions. To overcome this limitation branch prediction is employed and the most likely candidate is processed. As branch mispredictions necessitate that all prefetched data has to be discarded and reloaded, they incur high costs and should be reduced through suitable programming. An additional approach for hiding the potential stalls and which integrates well with the previous concepts is the out-of-order (OOO) execution. A scheduler dispatches instructions to ports, each of which serves specific functionality. The order in which the instructions are dispatched is not sequentially, but designed to minimize waiting times by tracking all dependencies and preferring instructions ready for execution. The final retirement is then carried out in the original order. The data and instruction dependencies should be taken into account when writing code, as tight dependencies can cause performance degradation. A core with the features stated above is given in a simplified form in Figure 4.2.

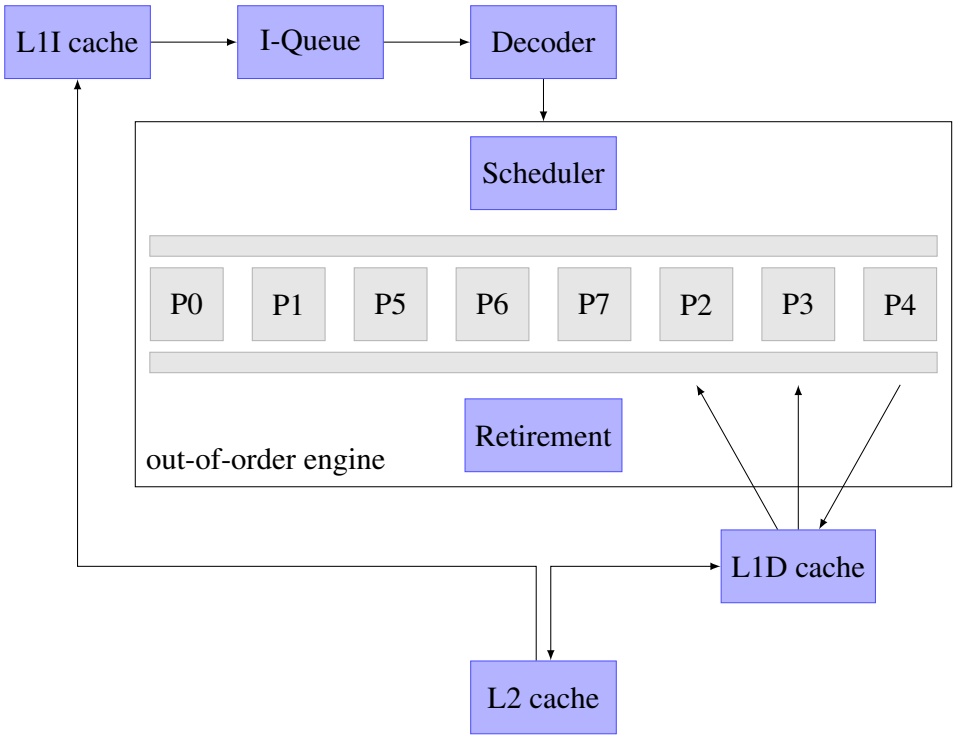


Figure 4.2 Simplified core configuration of the Intel Broadwell architecture based on [65].

A further big performance increase comes from the use of multi-core architectures. The arithmetic power usually increases linearly with the number of cores, not considering thermal dependencies. However, this is difficult to achieve in practice due to wide variety of reasons. The biggest issue is that the use of multiple cores not only has to be supported by the operating system, libraries and the software, it also influences the algorithms that can be used. This can be a severe drawback as even when everything is designed specifically for parallelization almost all algorithms incur an overhead compared to the serial counterparts. The other difficulty with multi-core system is the main memory bandwidth. While the lower cache levels are typically tied directly to a core, causing their bandwidth to scale accordingly, the higher levels and main memory generally do not. The quad-core version of the reference platform CPU offers four memory channels per socket, which is the same number as for the 14 core version and which therefore offers a lower bandwidth to arithmetic throughput ratio. This is the reason for an even

higher increase in the bandwidth gap [50]. An additional source of overhead associated with the used shared memory parallelization stems from keeping memory states consistent. This is handled transparent to the user, but has to be considered when designing multi-threaded code.

Through which hardware components the techniques introduced above are realized is outlined in the following. On the current Intel server platforms the cores within each CPU are linked by one or two bidirectional ring interconnect. Also attached to the ring are the hardware agent that handles communication to the periphery and the memory controller that manage the access to the main memory. On systems with many cores two rings are used which communicate with each other through the so called SBoxes. This setup is depicted in Figure 4.3. The L3-cache is shared between all cores and its data is distributed among all cache slices to maintain a uniform load. This is done through the ring interconnect. Multi-core architectures introduces some complications into caching, as all cached data must maintain consistency across all cores. When a cache line is held by multiple caches simultaneously significant latencies can arise from synchronization. It is managed using the MESI protocol. MESI stands for the flags that are set for every cache line, namely: modified, exclusive, shared, and invalid. Through the use of these flags each cache can determine whether a cache line can be used immediately or must be synchronized with an other core first. If not programmed efficiently a cache line may repeatedly transferred between cores, stalling the requesting core in the meantime. This can even occur if the data is modified exclusively per core, but resides in the same CL and is known as false sharing [11]. It can be circumvented by padding variables in critical regions to cache line size and coordinating memory accesses between cores.

The use of multi socket machines brings an additional level in the memory hierarchy. The main memory accesses are then distinguished between local, i.e. memory controller on the same socket, and remote on the other socket(s). This is known as non-uniform memory access (NUMA). For access to the remote memory the sockets communicate through what is called the QuickPath Interconnect (QPI) on Intel platforms and it is depicted in Figure 4.3. The previously introduced cache coherence also applies across multi socket systems. This makes it even more important to program in a cache coherent NUMA (ccNUMA) aware fashion [21], as synchronizations between sockets incur even higher penalties. In the cluster-on-die mode used here, a further level in the hierarchy is established by logically subdividing each CPU approximately according to the two ring interconnects and a single memory controller per domain.

This hierarchy further gives rise to different thread distribution strategies when only a subset of the available cores are used. The two most important distribution strategies are compact and scattered [67], which are also referred to as close and spread [116], respectively. The compact or close scheme aims to fill each smallest logical unit before assigning threads to the next unit. Here this unit are the NUMA domains, to each of which seven cores are tied. This strategy has the advantage of very fast inter thread communication and shared memory use since all threads of each NUMA domain use a common shared L3 cache. The major drawback however, is the inefficient use of the main memory interface when only some NUMA domains and their corresponding memory controllers are used. The scattered distribution strategy is the exact opposite, as the number of threads is balanced between all NUMA domains. This makes inter thread communication slower, but at the same time allows the full utilization of the memory bandwidth with only few active threads. As the bandwidth limitation is common case in scientific computing this is typically the preferred strategy and is also used in this work unless otherwise

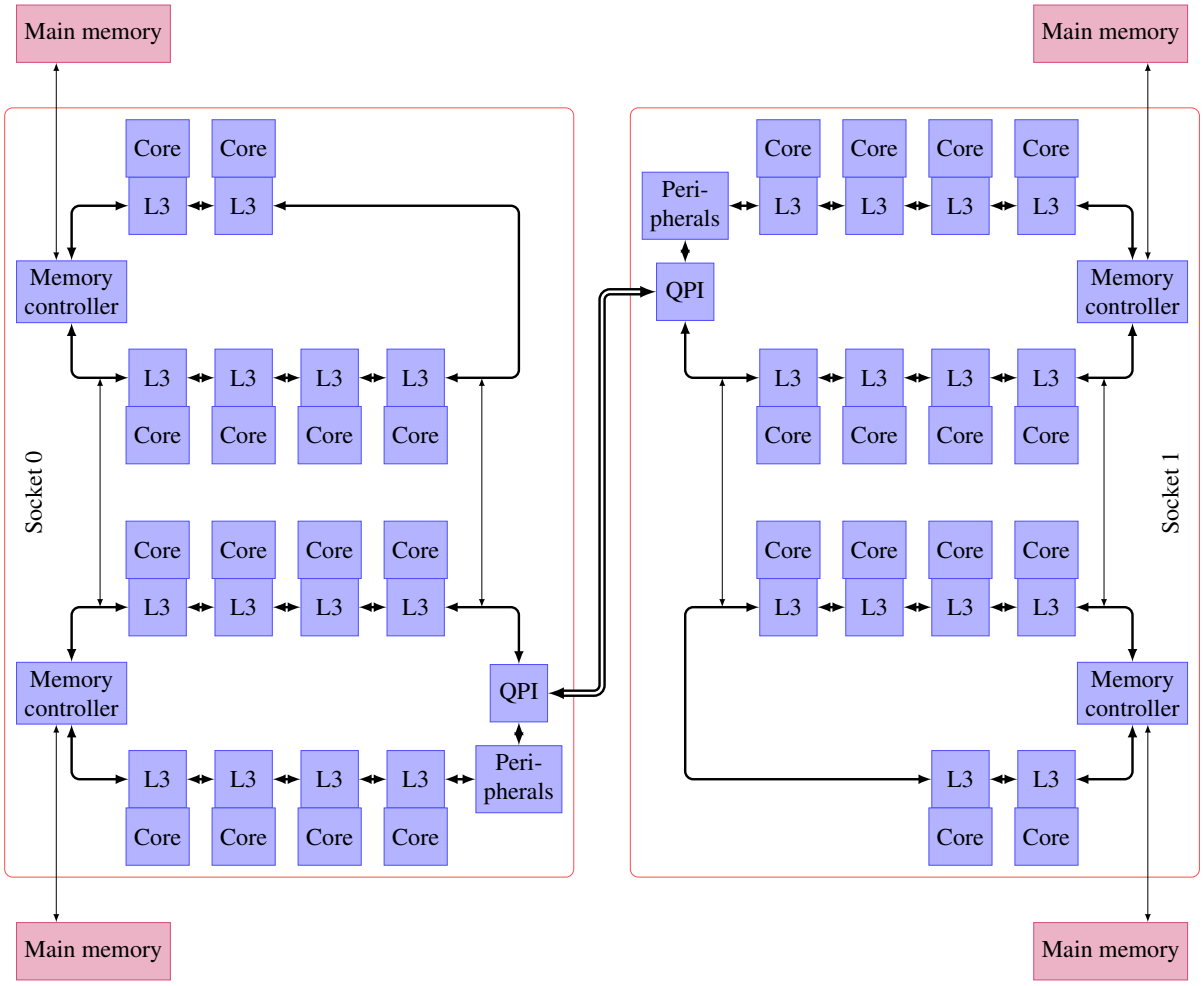


Figure 4.3 Simplified dual socket multi-core configuration as used for Intel Xeon E5-2690 v4 CPU systems. Derived from [64].

noted. In this context it should also be mentioned that it is generally advisable to bind (or pin) the threads to specific cores, as the system kernel by default carries out redistributions that can incur a large overhead. These stem from the cache that potentially has to be rebuilt on the new core, as well as the memory accesses across NUMA domains that can be required.

Apart from the cores and memory hierarchy there are some additional things to consider that do not directly relate to the bandwidth and compute limitation. The memory is addressed in userspace through virtual addressing. This allows programs and libraries to be loaded to arbitrary physical addresses and is the foundation of multitasking. The drawback is that every memory access requires translation from the virtual to the physical address. The mapping is controlled by the kernel, which maintains tables of allocated memory and the translations. This memory is organized in pages, where the size of a standard page on the reference platform is 4 KiB and the huge pages are 2 MiB and 1 GiB. The hardware offers support for automatically carrying out the conversion. To this end it maintains translation-lookaside-buffer, which store the mapping for a small number of recently used pages. There are multiple levels of this cache, which offers a compromise between speed and size. Maintaining a comprehensive list is very important, as every missing page initiates a page walk which requests the specific page from the kernel and is very costly. The limited number of pages that can be held in the TLB directly limits software in the number of outspread memory location which can be used efficiently.

4.1.2 Performance requirements and metrics

Taking the cross-section through all software in use today, a very wide range in requirements is raised. Designing separate hardware for every type of application is impossible, as is creating the one CPU that ideally satisfies all requirements. Hence, hardware manufacturers create hardware that performs well, but not optimal, for a wide range of uses. Much software also does not require highest performance, but is instead written for flexibility and security. This leads to most software only using a small fraction of the peak performance offered by the platform. This is different for scientific computing and simulations, where due to the controlled environment security is not of high importance. Instead performance plays a crucial role as problems and clusters can get very large and efficient resource usage offers large payoffs. But also in the field of scientific computing the hardware preferences can vary a lot, causing very problem dependent bottlenecks. Hence, even this type of software typically does not use the full potential and the benchmarks used for rating the top 500 supercomputers, which are tailored towards each system, only achieve between 50% and 93% of the peak performance [1]. A small subcategory are the streaming type codes, such as the LBM and FDM used in this work. They are characterized by a loop over many elements that dominates all computational cost. The loop kernel is mostly of lower complexity, but more importantly it has a mostly regular memory access pattern. These types of code are simple enough to be analyzed by hand with the models introduced in the Sections 4.2.1–4.2.3. These are based on the distinction between bandwidth and compute limited and are well suited for streaming type codes.

Some of the performance metrics used to assess code characteristics and which are supplied to the performance models can be measured directly. The most basic metrics consider duration or rates, but accurate time measurements in itself is already a complex task [155]. In this work the most relevant types of time measurements are the wall time and CPU time. The wall time is the actual time that has passed, just as an external clock (such as a wall clock) would measure and is

independent of computer internal timings. Complications such as leap seconds are not relevant here, but it should be noted that the wall time is only approximated on most computers and is subject to drift which could lead to negative durations in rare cases. The CPU time is instead monotonic and ensures positive time intervals as it essentially counts CPU cycles. The user time is a part of the CPU time and corresponds to the time an application was actually active on a core. This can differ a lot from the wall time as multitasking or waiting for resources can halt the execution of code and is thus omitted. Which timings are used depend on the goal of the measurement. Here the user time is used for the performance optimizations as it gives highly accurate readings for short intervals and a specific piece of code. For the test case comparison the wall time is more suitable as it represents the actual time a simulation takes, including all waiting times for resources and context switches. Apart from time measurements sophisticated analysis of the inner workings of modern CPUs is possible through built-in performance counters. These are very flexible registers that can be configured to count a wide range of events within the core or many other components in the CPU. By combining different events detailed performance metrics can be established.

In the following a set of units for the performance comparison and analysis is introduced. The comparison and the analysis have different requirements in granularity and further characteristics and are treated separately. For the comparison only the overall throughput is relevant, hence updates per second (UPD/s) is used. Here an updated refers to updates of individual grid points or lattice sites. Another common and equivalent term is in fact the lattice site update (LUP), but in order to avoid a too close association with the lattice-Boltzmann lattice the more neutral update is used. It should further be mentioned that the UPD/s are an adequate measure for the computational throughput important to optimization, but do not allow any conclusions to be drawn with regard to the actual simulation performance. Comparing the final simulation performance is an essential part of this work and is addressed in Chapter 5.

For the detailed performance analysis and modeling that is introduced in the following sections different units are required. The distinction into the arithmetic effort and bandwidth requirements govern the preferences. Starting with the arithmetic component, a measure needs to be used that can describe the hardware capabilities on the one hand and the software demands on the other hand. Ideally these two sides are fully independent through the choice of unit, e.g. the figure for the arithmetic potential of the hardware should not change depending on the software that is being run. In practice a compromise needs to be established and a discussion of the options is given in the following. The natural choice would be the use of mathematical operations directly, as they are readily available from the equations. However, the relationship between the equations that are being solved and the instructions a CPU eventually executes is very complex. There are many additional operations carried out that do not correspond to any mathematical operations at all, such as memory offset calculations or jump and branch instructions. Considering only floating point operations (FLOP) is therefore a sensible option. In fact the use of FLOP/s is very widespread, but on modern hardware the throughput of floating point operations varies greatly with the exact type of operation. On the one end of this spectrum are divisions, which take up to 16 cyc to execute on the reference platform. On the other end are vectorized fused-multiply add (FMA) operations. These carry out a multiplication and an addition on four separate sets of numbers in a single cycle, effectively leading to 8 FLOP/cyc. In conjunction with the two execution ports per core the throughput of the reference platform can therefore vary between 162.5 MFLOP/s and 41.6 GFLOP/s per core, depending on code. This makes FLOP/s a very good marketing tool,

but better options are available for the performance modeling. By choosing a unit that always corresponds to being executed in exactly one cycle the separation between code and hardware can be improved. Here, the instruction (INSTR) is used, as the reference CPU offers a throughput of 1 $\frac{\text{INSTR}}{\text{cyc}}$ per execution port for most floating point instructions, independent of vectorization or FMAs. The only exception relevant to this work are the divisions. To minimize the complication introduce by this dependency the operation is counted as multiple instructions, depending on their specific cost on the considered hardware. An additional potential obstacle is the limitation of the execution ports to specific operations. This can be circumvented by introducing a port balancing factor on the software side, but is not necessary for the reference platform as FMAs can execute on either port and thereby balances the use automatically. With the INSTR as the measure of choice no distinction between non-vectorized and vectorized instructions is made, since the execution duration generally does not differ. However, the difference is reflected in the bandwidth requirements. A suitable unit for the bandwidth is discussed in the following.

The choice of an appropriate data size unit is much more straight forward, as the size of a double precision floating point number is constant and used exclusively throughout this work. The size of a double is 64 bit, 8 B or one quadword (QWORD). The use of QWORD leads to ratio of one for one double per one instruction and which creates an intuitive understanding for a balance between arithmetic throughput and bandwidth requirement.

4.2 Performance modeling

In the following, three models for estimating code performance are introduced. They are based on the distinction between bandwidth and computational requirements and are given in an ascending order of complexity and accuracy. Their intended application is for compact loop kernel and for single core analysis, but they can be readily extended to multi core analysis.

4.2.1 Theoretical roofline model

The most basic approach and starting point for all other techniques is the roofline model calculated from purely theoretical values. This has the great advantage of allowing performance predictions before neither software nor hardware exists. At the same time the theoretical numbers can differ significantly from sustained performance [58], especially when those figures are also used in marketing. Nonetheless, it is an established model that offers great flexibility in the level of detail at which the model is used [16, 18, 161]. The roofline model as employed here has some minor adaptions compared to the original roofline model [161]. It is based on the distinction between computational and bandwidth requirements [130, 161], essentially reducing all hardware complexity to the core and main memory. It should be noted that the model can also be applied to the core and any cache level. Which approach is appropriate is a question of the analyzed loop kernel. Low loop iteration counts may be served entirely from a particular cache, while high counts tend to be served from main memory. This is demonstrated in Figure 4.4, which shows the throughput for the very simple “add” case of the STREAM [72] benchmark at varying loop lengths. The characteristic jumps in the throughput correspond to the individual cache sizes and speeds and match the predictions very well. In practice the domain size typically does not allow

all data to fit in the caches and most requests are served from main memory, which will also be assumed in the following.

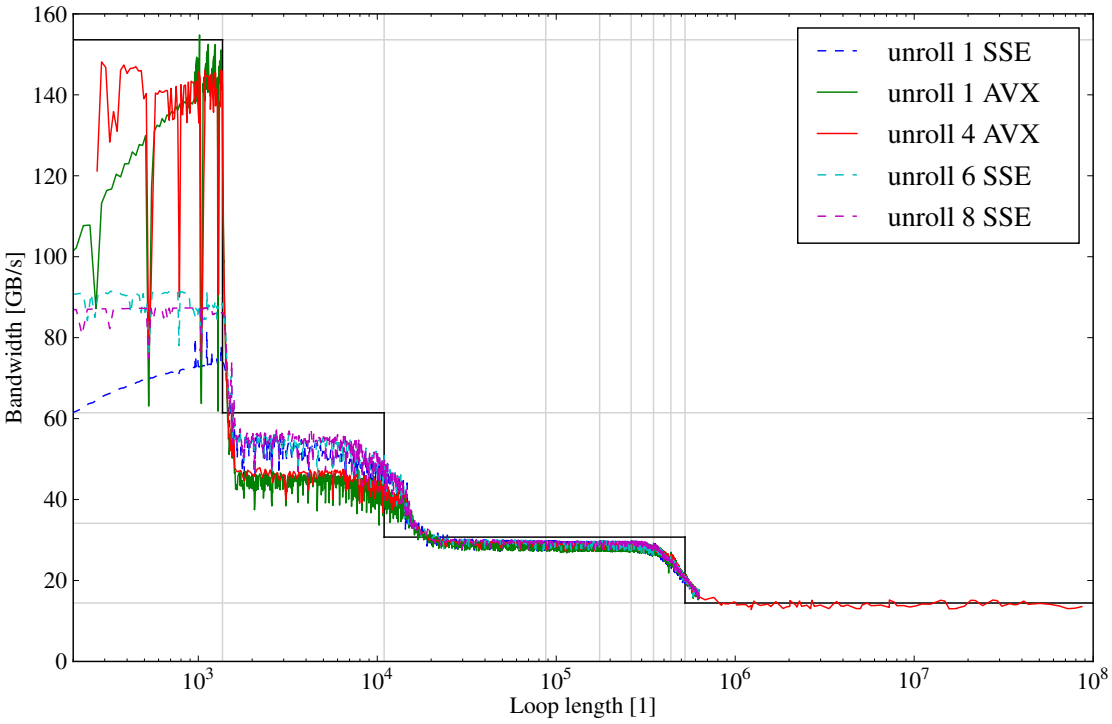


Figure 4.4 Measured and estimated bandwidth of the "add" version of the STREAM benchmark [72] (`A[]=B[]+C[]`) dependent on the array size. The various colors represent different tuning techniques to maximize the throughput. The solid black line is the estimate obtained through the roofline model, which is the result of the theoretical cache bandwidths (horizontal grey lines) and the cache sizes (vertical grey lines). The analysis was carried out on a Intel Xeon E5-1650.

For the roofline model the distinction into computational and bandwidth requirements of the code are combined as a ratio into the code balance B_c, while the hardwares computational and bandwidth capabilities are given as the machine balance B_m. These balances can for example be

$$B_c = \frac{\text{no. of doubles}}{\text{no. of instructions}} \qquad B_m = \frac{\text{quadwords per second}}{\text{instructions per second}}, \tag{4.1}$$

but the exact choice of units is flexible, as long as they match between both balances. A detailed discussion of possible unit choices was given in Section 4.1.2. Here, the operational intensity is expressed in floating point instructions, which helps in keeping the code and machine balance separated and is more accurate than using the arithmetic intensity that only takes mathematical operations into account [161]. Similarly the available bandwidth is expressed in quadwords, which is equivalent to the size of doubles and leads to balances of one when one double is transferred per instruction. This relationship between operations and data transfers means that the code balance by itself can already give an indication in which regard a code is computationally challenging. However, only the ratio between code and machine balance allow for a definite

answer. It establishes whether the code will be compute or bandwidth bound on that particular platform. This is determined according to the following criterion

$$\frac{B_m}{B_c} \begin{cases} < 1 & \text{bandwidth bound} \\ \approx 1 & \text{balanced} \\ > 1 & \text{compute limited} \end{cases} . \tag{4.2}$$

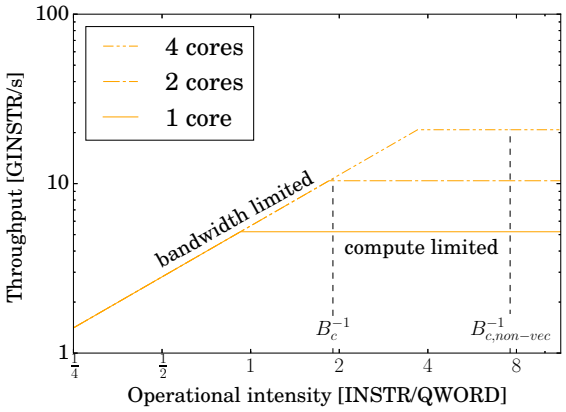
Once the bottleneck has been established the expected updates per second can be calculated from the limiting resource. For compute bound code/platform combinations the instructions per update and instructions per second are used and in the case of bandwidth boundedness the memory transfers per update and memory bandwidth are used. Note however, that due to the relatively large modeling errors that mostly stem from the use of purely theoretical values, this model only provides a tendency. Especially near the break even point the actual limiting aspect can influenced by a number of other effects.

The possibility to choose different characteristics for establishing the balances makes the roofline model very flexible in its application. This is used in Section 4.2.2 to further improve the prediction quality. However, even the theoretical version used here offers many choices that allow for the inclusion of many effects. The operational intensity for example could also include non-mathematical instructions and the occurring mathematical operation can be included by the actual (hardware dependent) cost. This is exploited by the choice of INSTR as the measurement for operational effort and for which a fused-multiply add is counted as a single operation and a division as 16 operations. Also the choice of the limiting bandwidth resource between the main memory or any cache level give the opportunity to include known caching effects. For small problem sizes it may be appropriate to a specific cache levels bandwidth. For the large problem sizes considered here the main memory bandwidth is the evident choice. Even then it gives the opportunity to include some caching effects. For example by eliminating data transfers that are known to be served by a lower cache level from the code balance.

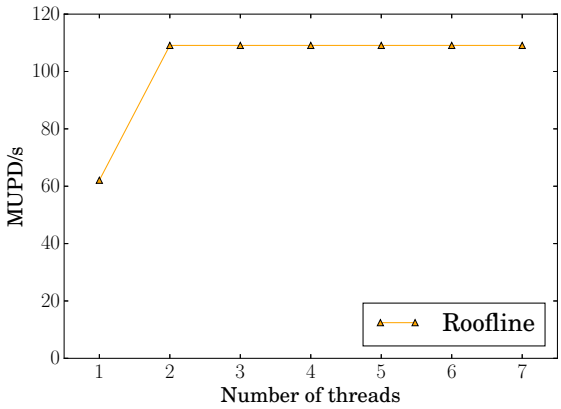
Extending the roofline model to multi-core systems is relatively straight forward. The original presentation of the roofline [161] is depicted in Figure 4.5a. The roofline represents the performance ceiling depending on a the code balance B_c and allows to differentiate whether the limiting factor is compute or bandwidth based. Multi-core configurations are included through multiple rooflines. This is impractical for the high core counts available on modern systems. Additionally the two simulation codes that are analyzed here have specific code balances and makes the presentation of the full code balance range superfluous. Instead the representation for a fixed B_c and a variable core count, as shown in Figure 4.5b-d, are used.

$$B_m = \frac{\text{memory bandwidth} \cdot n_d}{\text{single core throughput} \cdot n_c} = \frac{n_d}{n_c} B_{m,core}, \tag{4.3}$$

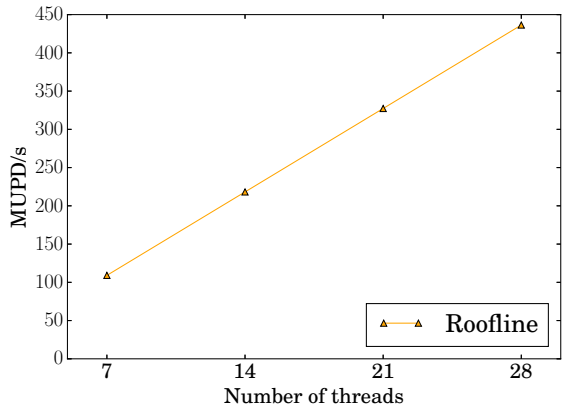
where n_c is the number of cores and n_d is the number of occupied NUMA domains. When the roofline model is applied to single NUMA domain multi-core systems the code balance B_c remains constant, since the loop kernel is not influenced by the number of cores. However, the machine balance B_m needs to be adapted. All resources within the cores, i.e. computational throughput and caches, are assumed to scale linearly in the number of cores as these resources are mostly independent. The main memory bandwidth on the other hand is taken as constant, since the number of memory channels stays the same and any advanced behavior, such as the



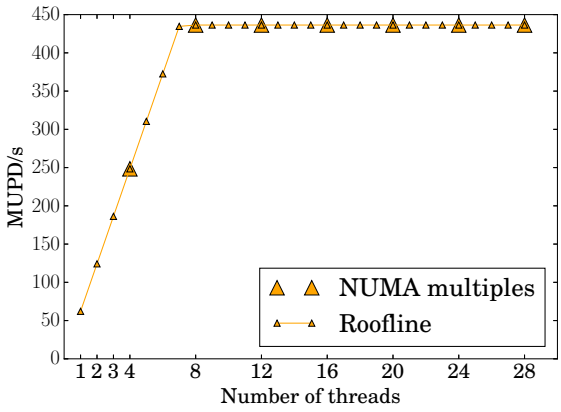
(a) Original roofline representation for single, double and quad-core configurations



(b) Compact thread affinity on a single NUMA domain



(c) Compact thread affinity per full NUMA domain



(d) Scattered thread affinity

Figure 4.5 The roofline model predictions for various affinities and representations.

shared L3 cache and memory interface saturation are only considered in the more advanced models. This leads to machine balance of Since the hardware used for this work has multiple NUMA domains the multi-core extension can be slightly more involved, depending on the thread affinity strategy. The compact scheme fills each NUMA domain before continuing with the next. For the first seven cores or first domain this is equivalent to a single NUMA machine as shown in Figure 4.5b. In terms of complete NUMA domains however the scaling of B_m becomes linear, as the available bandwidth as well as the arithmetic capabilities grow linearly. This is depicted in Figure 4.5c. When a scattered affinity pattern is used these effects are reversed, because the first threads are placed in individual domains, enabling the use of their respective bandwidths. Hence, on the reference platform up to the first four threads machine balance is constant and therefore the update rate scales linearly as demonstrated in Figure 4.5d. With any further addition of threads the machine balance decreases again just as for the single NUMA domain case, but at quadrupled core counts and update rates. The higher available bandwidth and the postponed bandwidth limitation in terms of absolute thread counts are the reason why the scattered thread distribution is predominantly used from here on.

In the following, the roofline model is applied to a basic version of the finite differences code, which does not include any of the advanced optimizations introduced later. It is the vectorized structure-of-array version that is described in detail within the tuning techniques in Section 4.3. On the one hand, this gives a baseline for the optimizations, on the other hand, it can be applied in a straight forward manner as no complicated schemes have to be modeled. The instruction count in Table 4.2 was extracted from the compiler output and it contains some optimizations. This is a more accurate and convenient approach than estimating the use of FMAs and further advanced instructions or than manually rearranging the equations, especially as the code is already available.

add/sub	mul	div	fma/s	min/max	unroll factor	IACA analysis [cyc]
112	44	6	65	18	4	183

Table 4.2 Instruction count and Intel Architecture Code Analyzer throughput estimate for a basic vectorized SOA version of the FDM loop kernel.

The instruction count given in Table 4.2 is used to calculate the code balance for the roofline model. For scalar code these counts correspond to a single update point. If however the code is vectorized, four values can be processed per instructions, as denoted by the unroll factor in Table 4.2. It follows that only a quarter of the cycles is required per update or alternatively four points can be updated simultaneously. All operations except the division have a throughput of one cycle in the vectorized version. The packed divisions require 16 cycles on the reference platform [33, 69]. Also required for the balance is the number of memory transfers per update. This can easily be determined, knowing that the stencil has a span of five and that the points in x-direction are perfectly positioned for being cached. This leads to nine grid points to be loaded per update. Additionally the right hand side (RHS) has to be read and stored, adding two more points. At every point there are four unknowns and the total number of doubles (or quadwords) therefore equates to 44 per update. This leads to the following code balance for the scalar version

$$B_{c,non\text{-}vec} = \frac{(9+1+1)\cdot 4}{112+44+65+18+6\cdot 16}\frac{\text{QWORD}}{\text{INSTR}} = 0.131\,\frac{\text{QWORD}}{\text{INSTR}} \tag{4.4}$$

and the vectorized version

$$B_c = \frac{(9+1+1)\cdot 4\cdot 4}{112+44+65+18+6\cdot 16}\frac{\text{QWORD}}{\text{INSTR}} = 0.525\,\frac{\text{QWORD}}{\text{INSTR}}. \tag{4.5}$$

The machine balance can be determined just as easily. The operations that can be evaluated per second are determined by the clock frequency at 2600 MHz and the number of execution ports that can handle the floating point instructions given above. On the reference platform these are port 0 and 1, each of which serves specific functions. To incorporate how well the instruction mix of the code matches the available ports, a port balance can be introduced. However, the FMAs can be processed by either port, which automatically balances their usage. The total main memory bandwidth of the reference platform is 153.6 GB/s, which results in 38.4 GB/s per NUMA domain and is equivalent to 4.8 GQWORD/s. The machine balance per core and the full system is then

$$B_{m,core} = \frac{4.8}{2\cdot 2.6}\frac{\text{GQWORD/s}}{\text{GINSTR/s}} = 0.923\,\frac{\text{QWORD}}{\text{INSTR}} \tag{4.6}$$

$$B_m = \frac{1}{28}\frac{19.2}{2\cdot 2.6}\frac{\text{GQWORD/s}}{\text{GINSTR/s}} = \frac{4}{28}B_{m,core} = 0.132\,\frac{\text{QWORD}}{\text{INSTR}} \tag{4.7}$$

The resulting ratio of

$$\frac{B_{m,core}}{B_c} = 1.758 \qquad \frac{B_m}{B_c} = 0.251 \tag{4.8}$$

means that the code on the entire platform is estimated to be bandwidth limited. In order to determine the expected update rate the limiting resource is used. Here the bandwidth limitation leads to

$$\frac{153.6}{44}\frac{\text{GB/s}}{\text{QWORD/UPD}}\,\frac{1}{8}\frac{\text{QWORD}}{\text{B}} = 436.4\,\frac{\text{MUPD}}{\text{s}}. \tag{4.9}$$

However, if only a single core is used, the code is limited by the peak compute performance and calculating the update rate from the performance figures above results in

$$2\cdot 2.6\frac{\text{GINSTR}}{\text{s}}\,\frac{4}{335}\frac{\text{UPD}}{\text{INSTR}} = 62.1\frac{\text{MUPD}}{\text{s}}. \tag{4.10}$$

Up to and including seven cores the code is estimated to scale linearly, at which point the bandwidth limitation sets in. This scaling is shown along with other thread distribution strategies in Figure 4.5.

4.2.2 Enhanced roofline model

The roofline model introduced in the previous section is a very versatile approach. It offers many ways to enhance the estimate quality. In the following some of these ways are shown for both major components of the roofline model, the code and the hardware side. The key to a higher quality estimate of a code's performance on a particular platform lies in the level of detail of the interactions between all the hardware features and the compiled code. There exist tools that model the behavior of the core in detail and can generate generally accurate predictions. In order to achieve this high accuracy, they are fed with compiled and compiler optimized code. This

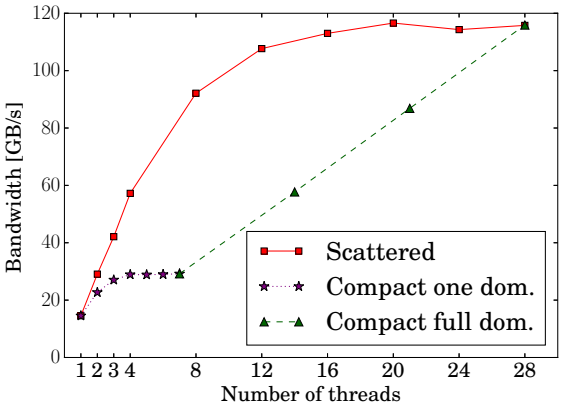
has the benefit of analyzing the code just as it will be used in production, but also the drawback that the code must have been written already or at least the relevant portions. There are several tools available for this analysis [14, 30, 66] with different levels of automation and granularity. The Intel Architecture Code Analyzer (IACA) is used here, as it offers the needed estimates and supports the Broadwell platform. Some of the advanced features accounted for are the number of registers, the dispatch to ports as micro-ops, overlap in the execution of instructions and the access and the transfers to and from the L1 cache. The analysis results can be found in Table 4.2 where the estimated number of cycles for the evaluation of the unrolled updates is given. The machine balance component to the roofline model is enhanced on the bandwidth side. It is important to note that the theoretical bandwidth cannot be achieved in practice and the gap to the sustained memory bandwidth can be significant. This has a multitude of causes, some of which are clock frequency differences between core, shared cache/ring interconnect and memory or latencies from the cache hierarchy and the distributed nature of the memory and further sources of overhead. The sustained memory bandwidth is therefore measured using a benchmark with minimal computational effort and high bandwidth requirements. The ADD case of the STREAM benchmark suite [72], which was also used to show the performance of different cache levels in Figure 4.4, is used. At the core to L1-cache level it has a load to store ratio of 2:1, which makes optimal use of the hardwares load and store ports. For the data transfers between the caches and main memory the load to store ratio increases to 3:1 due to the additional read for ownership. The vectorized add operation takes only one cycle to execute, which ensures a very high throughput and additional use of the multiplication port or both FMA units are not necessary. In Figure 4.6a the measured bandwidth is presented and it shows a clear dependence on the number of cores and thread distribution. The single memory controller is quickly saturated in the compact distribution, but the bandwidth scales linearly with number of NUMA domains. The scattered distribution exhibits a linear bandwidth increase with the first four threads and domains, before suffering from the same saturation as the single NUMA domain version, but on a higher level. One way to incorporate these findings and increase the accuracy of the roofline model could be by using the measured bandwidth per number of cores directly. However, in light of a different memory access pattern by the actual FDM and LBM codes, these are unlikely to be very accurate while simultaneously increasing the modeling effort beyond what is intended here. As ultimately the highest performance is to be achieved the measured bandwidth for 28 cores is used and the excellent scaling with NUMA domains allows exactly a quarter to be assigned to each domain. This simplifies the model and is consistent with establishing an upper bound.

Inserting the measurements presented above into the roofline model, a new code and machine balance are obtained:

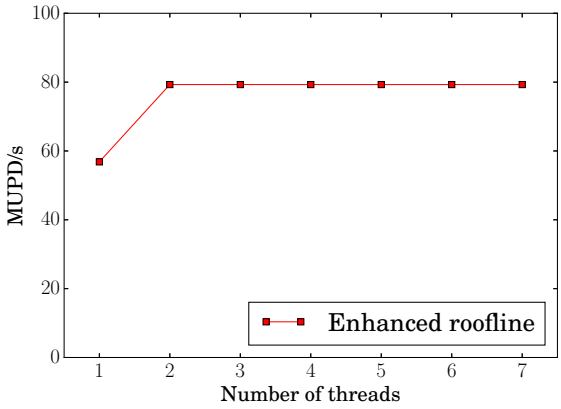
$$B_c = \frac{4 \cdot 44}{183} \frac{\text{QWORD}}{\text{cyc}} = 0.962 \frac{\text{QWORD}}{\text{cyc}}, \tag{4.11}$$

$$B_m = \frac{111.6}{28 \cdot 2.6} \frac{\text{GB/s}}{\text{Gcyc/s}} \frac{1}{8} \frac{\text{QWORD}}{\text{B}} = 0.192 \frac{\text{QWORD}}{\text{cyc}}. \tag{4.12}$$

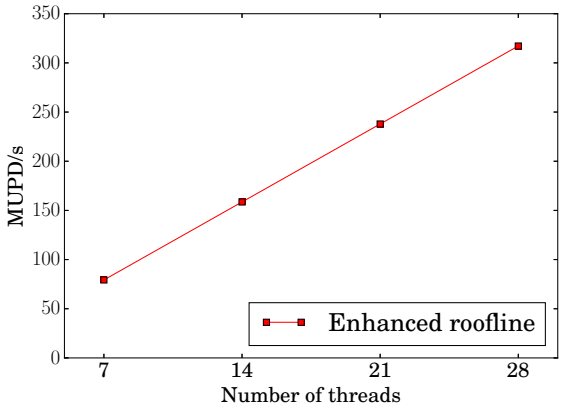
It should be noted that their interpretation is different to the code and machine balance given in the previous section and cannot be compared directly. Here the information on the port availability is included in the code analysis and not as part of the machine balance. The ratio of machine and



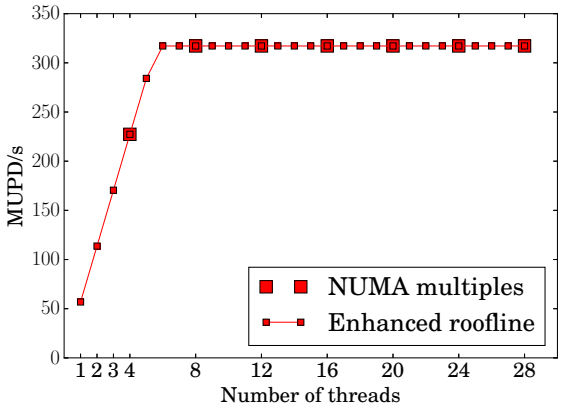
(a) Measured bandwidth for different thread affinities of the Add STREAM benchmark



(b) Throughput with compact thread affinity on a single NUMA domain



(c) Throughput with compact thread affinity per full NUMA domains



(d) Throughput with scattered thread affinity

Figure 4.6 Bandwidth of the Add STREAM benchmark and throughput predictions of the enhanced roofline model for various thread affinities.

code balance of the enhanced roofline model is then

$$\frac{B_m}{B_c} = 0.199, \tag{4.13}$$

which still results in a bandwidth bound case. Similarly the ratio of machine and code balance for a single core at 1.394 are still compute bound. Depending on the thread distribution the multi-core scaling shown in Figures 4.6b-d arises. The total update throughput at full hardware utilization and based on the bandwidth limitation is 317.2 MUPD/s. The single core throughput, governed by the peak computational performance, is 56.8 MUPD/s. Therefore at least six cores are required to achieve memory bandwidth saturation with scattered thread distribution. This can also be seen in Figure 4.6d.

4.2.3 Execution-cache-memory model

Further refinements to the models above are possible by additionally taking the cache hierarchy into account. This is especially important for highly tuned codes, which typically make excessive use of the caches. Within the plain roofline model the time spent on transfers between core and caches is not taken into account. This can lead to substantial errors. The execution-cache-memory model [49, 146] accounts for these effects as the caches are modeled as well. It has been used successfully for performance predictions and verification of code with varying levels of complexity [57, 88, 142]. In order to accurately incorporate the cache transfers into this model information on which data passes through which cache level is required. It can be established by using the layer criterion [124], measuring test runs with the help of performance counters, or by comparing to known behavior. The caches themselves are assumed to be ideal and no cache thrashing, false sharing or the limiting k-way nature for small ks is accounted for. Furthermore, the caches are assumed to exhibit no latency. In practice much of the latency is hidden through prefetching and other techniques, which makes this a reasonable simplification. The caches are however modeled with a finite bandwidth. Also included are some characteristics of the inter cache connections, such as possible overlap in communication, e.g. the main memory and L3 cache can exchange data at the simultaneously to the L2 and L1 cache. The possible single gatedness as well as half or full duplex communication of these links are taken into account as well. All calculations are based on the number of cycles that the transfer of entire cache lines between cache levels require. In the following these timings are denoted as $T_{a,b}$, where a and b can be any of the cache levels 1, 2, 3, main memory M, or the core C. Furthermore, the load and store directions between L1 cache and core are separated, because their bandwidth can be asymmetric and concurrent. The duration of arithmetic operations are given as T_{core}. The specific configuration is derived from the hardware documentation as far as available. The remaining parameters are determined experimentally. It should be noted that there can be discrepancies between the figures provided by the manufacturer and the measured sustainable performance. There are a multitude of potential sources of these discrepancies and an investigation into these is far beyond the scope of this work. For the Broadwell architecture the caches have been determined to act fully non-overlapping within themselves, but overlap with the core's computational cycles [49, 56]. Hence the total number of cycles per update of a CL is

$$T_{total} = \max\left(T_{M3} + T_{32} + T_{21} + \max\left(T_{1Cload}, T_{1Cstore}\right), T_{core}\right). \tag{4.14}$$

Transfer level	MM-L3	L3-L2	L2-L1	L1-core load	L1-core store
Cycles per CL transfer	7.934	2	1	1	2

Table 4.3 The number of cycles required to transfer one cache line between two caches, main memory or the core on the reference platform. The connection between L1 cache and the core is asymmetric.

The transfer timings between the different cache levels summarized in Table 4.3 are derived from the hardware configuration given in Table 4.1. It has been suggested that the effective bandwidths may be lower [58]. The model can be enhanced further to explicitly incorporate the penalty of clock frequency changes at the transition from L2 to L3 cache [56]. Using too many penalty parameters makes the model unnecessary complex and can lead to fitting the model to the desired outcome. This would of course completely defeat its purpose and any predictive capabilities. In the ECM model configuration used in this work only a single parameter is fitted to measurements. The main memory to L3 cache transfer speeds are recovered by modeling the ADD case of the STREAM benchmark and deriving the missing main memory to L3 cache transfer rate from the measurements shown in Figure 4.6a. This process yields the 7.934 cycles per CL transfer listed in Table 4.3. The advantage of modeling and solving for the missing transfer rates rather than directly using the measured bandwidth and clock frequency is that some of the additional penalties are indirectly included.

The core's computational effort can be determined relatively accurately by using the tool assisted analysis introduced in Section 4.2.2. Apart from the arithmetic throughput it also yields the L1-cache to core communication behavior. The tool assisted generation of the update duration T_{IACA} therefore not only replaces the manually established computational effort estimate, but also a part of the L1 cache bandwidth usage. The result is the model

$$T_{total} = \max\left(T_{M3} + T_{32} + T_{21} + T_{ld/st,IACA}\,,\; T_{IACA}\right). \tag{4.15}$$

In the case of bandwidth limitation it can be simplified to the sum of the number of cycles per cache level

$$T_{total,BW} = T_{M3} + T_{32} + T_{21} + T_{ld/st,IACA}. \tag{4.16}$$

It should be noted, that all calculations are based on full cache lines, as this is the smallest unit at which caches operate.

By applying the ECM model to the basic example used previously, a more accurate estimate can be obtained. The cache usage is determined through the layer/line criterion. For this the number of updates until the same point is encountered again are counted. These encounters occur once every line and every layer until the entire stencil has passed. From the number of updates the cache with sufficient size can be determined. This is straight forward for the points on the stencil in x-direction since they are reused in subsequent steps, except for the first one. Since the cached data is reused immediately the data will still reside in the L1-cache. This is also the premise for the blocking technique introduced in Section 4.3.5 and the lines and layers are visualized in Figure 4.10a. For ease of modeling all other points are ensured to be loaded from main memory. This is achieved by the domain shape as a single long strip in x-direction and which means that each point is only visited once per full domain update. The number of transferred doubles per cache level are summarized in Table 4.4, separated into load and store operations and by the

source and target field. Given along with it are the number of cycles required for the transfers between each level.

	MM-L3	L3-L2	L2-L1	L1-core	computations
read/load [CL]	9u+1r	9u+1r	9u+1r	13u+1r + spill	
write/evict/store [CL]	1r	1r	1r	1r + spill	
duration [cyc]	349.1	88	44	305	366

Table 4.4 Number of data transfers and duration between cache levels for a plain SoA implementation of the finite difference stencil. The "u" represents four unknowns, while "r" represents the four right hand side values and "spill" covers all values that have to are written back to the L1 cache as the result of register spill as determined by IACA.

The durations are straight forward to determine from the transfer sizes and cache bandwidth provided in Table 4.3. Only the computation and L1-core transfer duration have been taken from the IACA results directly. The total duration of updating eight points or one cache line according to equation (4.15) is 786.1 cycles or conversely 26.5 MUPD/s.

Within the ECM model multi core scaling follows the same approach as with the roofline models [49]. All per core resources are assumed to scale linearly with the number of cores. For most components this assumption is apparent, as all core resources, L1 and L2 caches are duplicated and almost fully independent. However, it also includes the shared L3 cache since its distributed nature causes the bandwidth to scale with the number of cores. The slight increase in the average distance between the L3 cache slices is small and mostly impacts latency. Additionally, there are many unpublished properties of the L3 cache behavior, which makes more detailed modeling problematic. As the only resource the main memory bandwidth is considered constant. In order to keep modeling simple, the single core performance determined with the ECM model is assumed to scale linearly until the memory bandwidth barrier is reached. This limit is known from the previous models. An alternative approach would be keeping the total main memory transfer duration across all threads constant and scaling everything from the L3 cache down. This approaches the same result as the aforementioned technique. Additionally the stall cycles induced by the main memory bottleneck can be used for other cache transfers, effectively increasing their throughput. This again causes only the main memory bandwidth to be the limiting resource and leads back to the original simple approach. The ECM offers enough flexibility to model even more effects and potentially increase accuracy. However, for most applications this is not necessary and the effort of modeling increases considerably. However, for the detailed analysis carried out in Section 4.4.2 of the source of model and measurement discrepancies, several additions are incorporated.

In Figure 4.7 a comparison of the three models and the actual measurements are shown. It can clearly been seen that the higher detail models perform better. However, even the ECM model does not capture the characteristics of the actual loop kernel. These discrepancies stem from the very complex stencil evaluation and are investigated in more detail in Section 4.4.2. The introduction into the performance modeling techniques given here is only basic and meant to facilitate understanding of the optimization techniques and the interaction between tuning and modeling presented in the following section. It is also meant as a primer to the modeling of the even more complex, fully optimized stencil evaluated in Section 4.4.1.

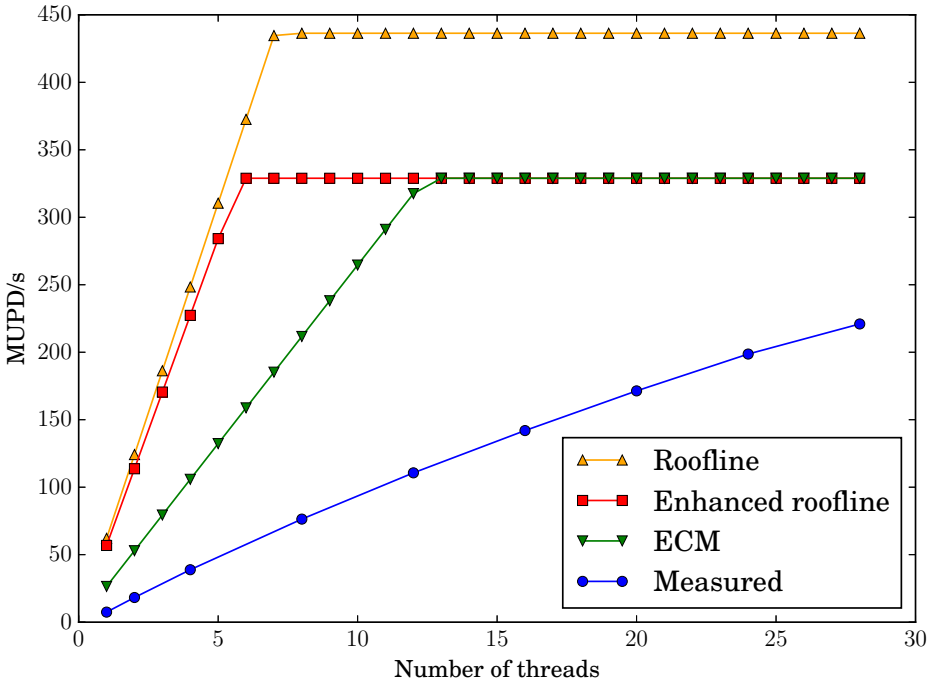


Figure 4.7 Comparison of the measured performance with estimates created with the three performance models for a simple implementation of the FDM loop kernel.

4.3 Optimization techniques

In the following, several different tuning techniques are presented. They target programming strategies and configuration, but leave the mathematical aspects of the solver intact. Of course, the performance of the method and algorithms should already be optimal as later changes will likely eliminate all code optimization gains. Also, improving the convergence and reducing the algorithmic complexity can potentially reduce runtime by orders of magnitude. The number of iterations or of mathematical operations and smaller storage and bandwidth requirements also have a very large impact on performance which is typically larger than the payoff from code tuning. Here the methods have been established already and the significant gains from the tuning techniques presented in the following are required to be able to compare the highest possible performance.

Efficiently using modern hardware is not straight forward and implementing scientific code directly from the equations generally yields suboptimal code. To better use the hardware features usually some of the readability and flexibility have to be sacrificed, but as described in Section 4.1 these advanced features are essential to high performance. Conversely a misuse of said features can cause drastic slowdowns. A careful selection is therefore needed. As presented in Section 4.2, bottlenecks can typically be split into the two categories of compute and bandwidth limited. Many of the techniques shown here address either of these issues directly, but due to the complexity of the subject some approaches only yield gains in conjunction. In order not to sacrifice all flexibility and portability some of the very exotic measures are not explored here. The field of lattice-Boltzmann methods has seen some very invasive optimizations [5, 163] that are not

implemented on this account. For the presented steps the focus lies in gradually improving performance in modular ways and retaining as much flexibility as possible. Also the cross platform compatibility should be retained as much as feasible and therefore the compiler is guided to produce the desired code instead of writing in assembly directly.

The performance optimization steps lend themselves well to be embedded in an iterative process. First a performance model is used to determine where performance gains can be achieved and how much can be expected. It helps determining the bottlenecks and appropriate techniques to overcome these. The improved version is then verified against the performance model again. When discrepancies arise there are two potential ways to resolve these. Either the model is inaccurate, in that some assumptions were invalid or it did not take into account some advanced architectural feature. Alternatively the tuning technique did not address the original issue in the expected manner. In any case the cause must be determined and rectified before further optimizations are applied, either by adjusting the model or altering the optimization. Unfortunately performance models tend to decrease in accuracy with increasing tuning complexity. This can make it difficult to determine the true cause of any deviations. The successive degradation of estimate quality is analyzed in Section 4.4.2. As the iterative procedure is a very lengthy process the performance estimates are not applied to every step in the following. In Section 4.4.1 the previously introduced models are applied to the final product instead.

The tuning techniques are demonstrated on the basis of the FDM and LBM code introduced in Section 3. The finite difference version starts out with a direct and unaltered implementation of the discretized weakly compressible Navier-Stokes equations as given in Section 3.2 equations (3.34–3.39). It was written with performance in mind, but without the application of advanced techniques. Hence the base version, denoted as I in Figure 4.8, is a serial version using a monolithic kernel and is compiled with a recent compiler (GCC 6.3.0) on high optimization levels. The memory layout is a structure of arrays (SoA) layout to facilitate later tuning steps and is scrutinized in Section 4.3.2. The initial version of the lattice-Boltzmann code is not as basic as the initial FDM implementation since it originates from an extensive tuning project. However, many of the advanced features can be configured or altered to behave just as a straight forward implementation would. This version is also denoted as I in the following. The most important difference to the FDM code in regards to tuning is the use of indirect addressing, which allows complex domain layouts and BCs to be handled transparently, as the neighbor cell addresses are looked up in a dedicated array. However, this inhibits the application of some of the techniques introduced later, while others occur naturally. If no equivalent technique for the LBM is available it is skipped and left out of Figure 4.8. It is important to note that not all memory accesses are indirect. Rather only the pull operation is indirect, while the post collision write operations are directly to the SoA memory layout. This enable some equivalent optimizations to the FDM version. For the measurements both codes are used in a separated, stripped down benchmarking version. These only contain the main loop over the domain and of course the loop kernel. This allows more precise measurements, but does not take away from the relevance to the full codes as it typically dominates the runtime. For the finite difference code all boundary conditions were simply disregarded, which for the chosen storage layout naturally results in a form of a skewed periodic boundary. Due to the indirect addressing the true bounce back condition can be applied without penalty for the LBM code. This avoids invalid values due to reads outside the actual domain and which could result in “not a number” (NaN) and denormal numbers, which are slower to process and could influence the measurements.

The concept of modular and incremental performance improvements made clear in Figure 4.8, which gives an overview over the speedup after every step. A detailed explanation to every step is given in the following. In section 4.3.9 this is followed up by running the same set of steps on different hardware architectures to determine the portability and the influence of architecture changes on the different techniques.

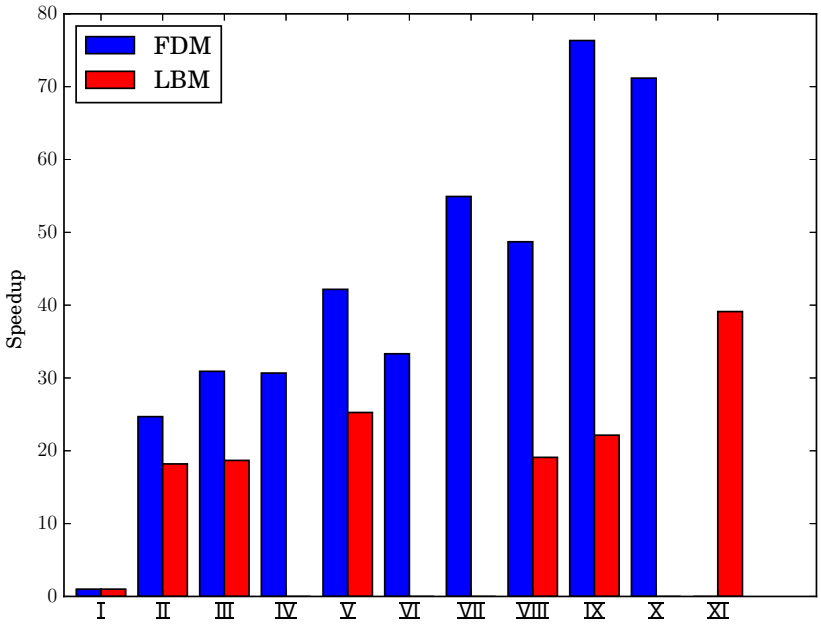


Figure 4.8 Speedups of the FDM and LBM code for the different tuning stages: I serial, II parallel, III vectorized, IV AoSoA memory layout, V semi-stencil, VI spatial blocking, VII semi-stencil and spatial blocking, VIII NUMA aware spatial blocking, IX NUMA aware semi-stencil and spatial blocking, X edge based evaluation, XI non-temporal stores

4.3.1 Multi core

Unsurprisingly the largest performance gains come from the use of multiple cores. On the reference platform 28 cores are available. However, current systems with Intel's general purpose CPUs support up to 28 cores per socket resulting in nodes with up to 224 cores and 448 threads [70]. In practice the full theoretical performance increase cannot be achieved, as many sources of overhead come into play at such large core counts. The biggest difficulties arise when the algorithms themselves do not scale well. Most come with some parallelization penalty and only very few scale perfectly, which then generally fall into the category of embarrassingly parallel, i.e. requiring no interaction or shared resources. In practice however, even embarrassingly parallel problems do not scale optimally as not all resources on multi core platforms scale with the number of cores or thermal management becomes an issue. The most important limitation in this regard is the main memory bandwidth, as determined in Section 4.1. A further factor introduced in the same section is the cache management overhead for ensuring cache consistency. This

overhead can increase severely for very large shared memory platforms. A final limitation are the inevitable small serial parts of the code which are unavoidable in almost all circumstances and which increase in importance with the number of cores [3]. Apart from practical scaling limitations of massively parallel codes, there is also the drawback of more complex and sensitive code. Not only does the load balancing and thread synchronization have to be integrated, but also efficient cache and memory handling for the distributed accesses. For example, code that does not take into account the NUMA domains can result in considerable performance penalties. The exact impact is analyzed later, accompanied by optimization steps dedicated to this issue.

The actually achieved speedup for the FDM and LBM code are shown in Figure 4.8 as Ⅱ. They are a factor of 24.7 and 18.2 for the FDM and LBM respectively, which is good considering it is achieved with a basic code version that does not account for the parallelization pitfalls given above. However, it has to be carefully assessed why this high speedup is achieved, as good scaling does not necessarily mean good performance. Quite the contrary can be the case since a code also scales very well when it is highly computationally intensive due to inefficient arithmetic operations.

4.3.2 Vectorization

In order to overcome potential arithmetic throughput limitations the codes are vectorized. This effectively quadruples the computational throughput as SIMD scales perfectly when certain prerequisites are met. The most important is a structure of arrays memory layout, which orders data by the field they belong to first and then the coordinates [151] rather than the intuitive other way around. This way the same component of neighboring points are adjacent in memory, which allows both points to be processed in parallel by SIMD operations. Both codes make use of the SoA layout, specifically with vectorization in mind. There are some drawbacks to the SoA layout. First of all it is counter intuitive which can make the code harder to read and more error prone. More importantly however, it can lead to other performance problems. This stems from the fact that the data required for a single update is more spread out in memory than with an array of structure (AoS) layout. This is unfavorable in regard to data locality concepts that the caches rely on. Furthermore, it can overstrain the translation lookaside buffer, which can only hold a limited number of memory pages at once. On the Broadwell platform this limit is at 32 huge pages. Counting the number of simultaneous data streams this number is clearly exceeded. For a sufficiently large domain the central point of the FD stencil alone requires access to four distant memory locations due to the four unknowns. The neighboring points in x-direction are included in these four pages as they are adjacent in memory. For the y and z-direction however the memory locations are again placed far apart and each point in turn has four unknowns resulting in additional 32 separated memory locations. The memory access pattern for the unknowns is visualized in Figure 4.9. Reading and writing the RHS adds an additional four page table entries and there are also additional pages for general use, such as the stack and other temporary variables. All in all the number of concurrently used data streams sums up to a total of over 40. The result are frequent first level TLB misses. The second level TLB handles most of these, but if it cannot serve them then costly page walks ensue. Then the virtual to physical address translation has to be requested from the Linux kernel, which manages the pagetables. A technique to circumvent these drawbacks is introduced later. For vectorization there are further prerequisites that have to be met apart from a suitable memory layout. All vector read and write operations

must be independent from the operations on neighboring points within the SIMD width. An operation such as `A[i] = A[i] + B[i+1]` cannot be directly vectorized as the result would depend on the iteration direction of `i` when `A` and `B` alias, i.e. reference the same memory. Of course with code written in assembler language or the more versatile intrinsics all knowledge of data dependencies can be directly taken into account by the programmer. However, to retain more flexibility, regular C code is used and the compiler is guided to generate the desired code. This can be achieved through standardized keywords such as `restrict`, which in the previous example would guarantee the compiler non-aliasing arrays. As these keywords do not give full control over compilers, there are specific directives that can be used to hint memory address alignment or data independence. Naturally, this sacrifices some portability between compilers. As maintaining alignment is particular important for later optimizations it is discussed more closely there.

The performance gain achieved through vectorization, denoted as Ⅲ, for the FDM is 26%, which is only a fraction of the theoretical 400% increase in computational power. This could lead to the conclusion that Ⅱ is at the boundary between compute and bandwidth bound and the code is now in a purely bandwidth limited regime. However, there are several other inefficiencies that limit the performance gain. The analysis with the performance models in Figure 4.7 shows a gradual change from scaling like compute bound code to bandwidth limited behavior. This is indicative of multiple limiting factors, which spreads out the transition. These issues are addressed in further steps which then enable the vectorization to deliver it's full potential. For the LBM no gains from vectorization can be observed. This is caused by the indirect addressing used in the pull operation, which is not suitable for vectorization as the arbitrary nature of subsequent memory accesses require additional operations. The data would have to be loaded in a scalar fashion first and gathered into the correct place in a register. This causes significant overhead for which the compiler heuristics assess the payoff and decide on the strategy. In the context of the coordinate splitting scheme a remedy is provided that allows for vectorization of most of the loop kernel.

4.3.3 Array of structures of arrays memory layout

In this work the array of structures of arrays memory layout is used only with the FDM code. It is also possible to use with the LBM as well, but the indirect addressing scheme renders this superfluous. The AoSoA layout addresses the potential TLB issues that arise with the SoA layout, but it is mostly a technique that enables further optimizations later on. As the name implies the AoSoA layout concatenates small SoA sections. Small groups of points are stored in the SIMD compatible SoA format, but these groups are stored consecutively by coordinate position as depicted in Figure 4.9. This keeps the memory addresses for all values belonging to a single point close and almost always on a single memory page, which greatly reduces the number of simultaneously accessed pages from 40 to 10 [151]. The AoSoA layout not only increases the data locality which reduces TLB pressure and improves caching behavior, but it also serves a second purpose. The block size that specifies the SoA block size within the AoSoA layout integrates well with spatial blocking techniques introduced later. It greatly simplifies the implementation when matching block sizes are used. As mentioned at the beginning this layout change mostly enables other techniques and it is therefore not surprising that the measurements shown as Ⅳ in Figure 4.8 do not indicate any performance increase. Also the benefits of the

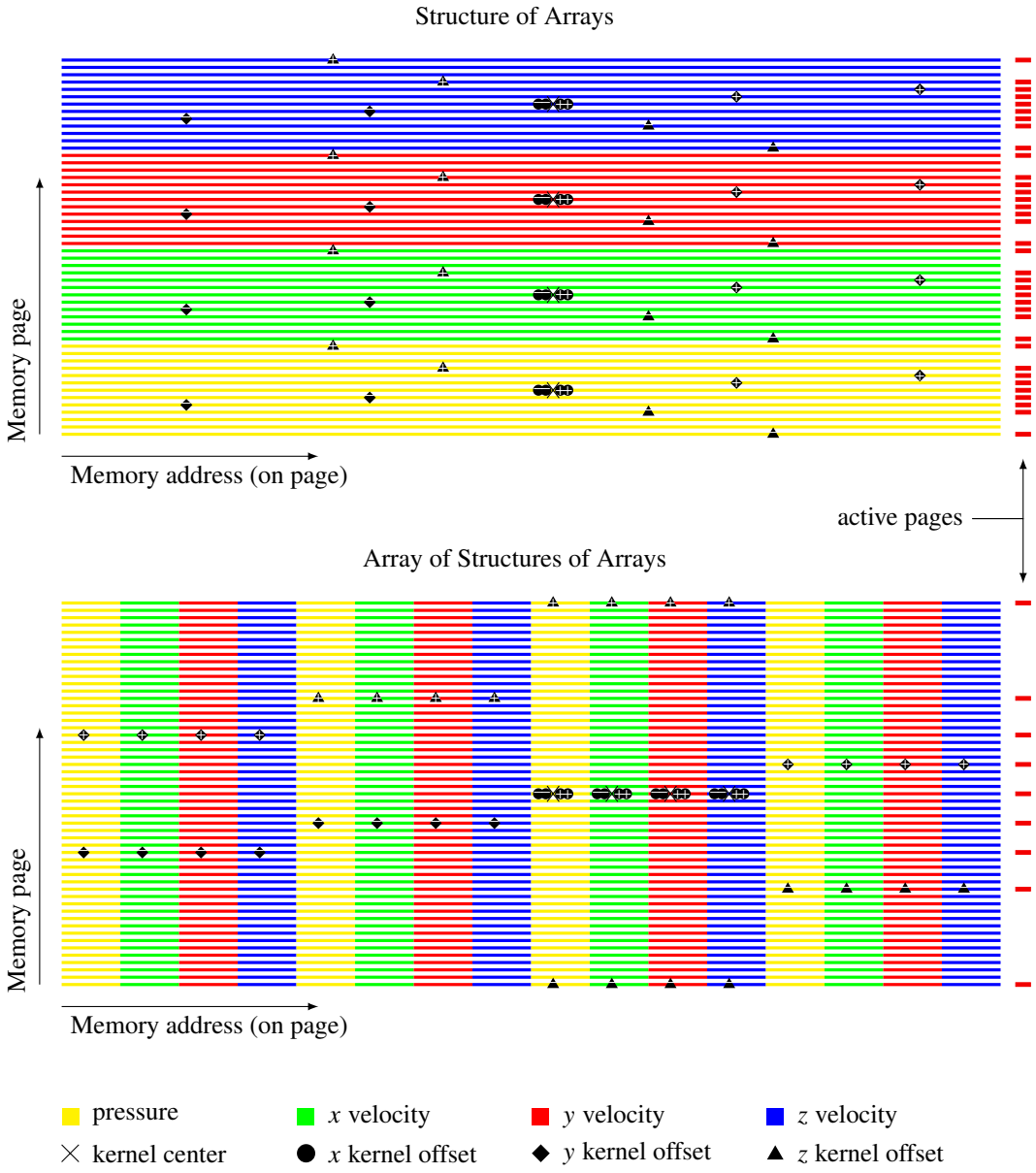


Figure 4.9 SoA and AoSoA memory layout and the memory access pattern for a single FD stencil evaluation.

AoSoA layout are concealed by other performance bottlenecks. It should be noted that there are also other techniques that enable vectorization with improved data locality. For example vector folding [168] is such an approach, but these complex techniques make integration with further optimizations difficult or even impossible, resulting in a slower overall performance.

4.3.4 Semi-stencil

From the previous steps it is clear that some fundamental issue in the FDM code limits the impact that the tuning techniques had. This problem lies in the complexity of the loop kernel. The many terms and the fourth order central differences result in a very long code with many intermediate values. The architectures supporting the AVX2 instruction set provide 16 registers which are by far to few to keep all values in the core. Instead intermediate values must be written back to memory to free up registers which is commonly referred to as register spill. As the spilled values are local to every core and will be loaded back into registers within a single iteration they are usually handled by the L1 cache only. As this cache is very fast the performance hit is dampened. However, the large number of spills lead to significant performance hits. Through the automated analysis of the kernel in section 4.2.2 it is known that 247 loads are performed per iteration, 56 of which are theoretically sufficient. This means that 77% of all loads are due to register spills. This is even more significant for stores, of which 92% are caused by spills.

A technique to reduce the number of intermediate values and hence the register pressure is the use of semi-stencils or coordinate splitting [28]. It is based on the idea of separating the calculation of the differences required for a full stencil evaluation into the coordinate directions and processing each in separate consecutive loops. A more general term for this is semi stencil as splitting the kernel is not limited to coordinate directions. However, it is a convenient choice for the FDM stencil used in this work.

The splitting is realized by moving the evaluation of terms with the differing difference directions to separate loops that add their intermediate results to a buffer. A final loop processes the buffer and updates the right hand side, completing the stencil evaluation. Due to the equations used in this work there is always some overhead involved as the directional components of some terms are not independent. This gives two options for the type of data to place in the buffer. One approach is to store all raw values that is needed in the following loops. This results in a total of 14 buffered values. The alternative is to calculate the contribution per coordinate direction to the primary variables directly. This requires only 4 buffered values, but some computations need to be repeated in every loop. In Figure 4.8 the second approach is presented as ∇ as the first version was 6.6% slower in tests. The overall speedup achieved through coordinate splitting is 38%, despite an overall increase of arithmetic instructions and more designed load/store operations. However, the instruction mix is essentially adjusted to better fit the hardware capability. This means a higher number of arithmetic operations but decreased numbers of actual loads and stores due to the reduced register spill.

The same technique can also be applied to the LBM by splitting into a semi-stencil. This is the more appropriate term in this context as the loop is split in density distribution function pairs rather than coordinate directions. Also the considerations for splitting the stencil are slightly different, as the split allows the part with the indirect addressing to be split from the others. This way most parts can be vectorized efficiently. The critical and first part in this operation is the task of loading all distribution functions from the neighboring cells into a buffer, which is essentially the pull

operation. Also part of this first loop is the calculation of common components required for the following collision operations. Since the collision relaxes towards an equilibrium distribution most equilibrium contributions are predetermined in this way. The nine loops following the pull part extend the common components by adding the directional terms. This can be done with SIMD instructions as the buffer and the target fields are in SoA memory layout and all entries are processed sequentially. This approach results in a 38.8% speedup.

4.3.5 Blocking

The blocking approach introduced in the following is also known as loop tiling [27, 124, 165]. It is a method to improve cache usage and thus takes pressure of the memory interface. This is achieved purely by reordering the iteration sequence over the domain, as shown in Figure 4.10b. It requires no modification of algorithms or data layout. The reasons for and design of the blocking techniques differ for the LBM and FDM and are therefore provided separately. Analyzing the cache use of a regular FD stencil loop over a sufficiently large domain shows that generally only the points in x-direction make use of the cache. The first point at $+2x$ is loaded from main memory and is then reused in the next iteration as $+x$ and so forth. For very short domains this can also work for points in y-direction, given that the domain length to cache size ratio is small enough. To create a universal solution without dependence on the actual domain size the iteration sequence can be altered to only process a limited number of points in x-direction at once, which creates small strips of domain. In conjunction with the AoSoA memory layout a block size in x-direction naturally arises. When these memory blocks are not processed in x-direction, but in y-direction first, a very simple blocking is achieved. With a suitable choice of the block size in x-direction the already cached unknowns in y-direction can be kept in the cache for longer, thus ideally reducing the number of main memory accesses from 44 to 28 or by 36%. Of course some overhead arises from the edge cases, but these can be minimized by maximizing the block size to the limits of the caches. Continuing the idea of splitting the unit stride direction in blocks and processing them in the second or y-direction, the second direction can be blocked as well. This enables the unknowns in z-direction to be cached also. Figure 4.10a shows such a single block and the cache use under ideal conditions. Blocking the second direction further reduces the number of unknowns that must be loaded from main memory to 12 per update, which is a reduction by 73% compared to the fully non-blocked version. Simultaneously it results in an increased overhead from the edge and corner cases, but it is small compared to the overall reduction in main memory accesses. The choice of the x and y block sizes is bound to the cache sizes. They are chosen in such a manner that the points in y direction can be loaded from the L2 cache, while the points in z direction are provided by each core's portion of the L3 cache, as shown in Figure 4.10a. Certain block sizes should be avoided, as the cache associativity can result in self interference [92], which in turn has to be circumvented by padding [119]. A further enhancement lies in the distribution of the individual blocks between threads and the order in which they are processed. Looking at the grid of blocks that is formed for a linear distribution slender strips belonging to each core are produced, except in special circumstances. These strips result in large surfaces between thread regions. Due to the stencil width this requires frequent memory access to other threads memory regions that are potentially on a remote socket. In order to generate thread regions that are as compact as possible the distribution is performed along a

Hilbert curve as demonstrated in Figure 4.10b. This provides a generic way of keeping the blocks of a region close and thus reduces the shared surface between regions.

The ideal block sizes maximize the cache usage. In the following these block sizes are calculated for the combination of the fully optimized FD loop kernel and an ideal cache structured analog to the reference hardware. It should be noted that in practice caches are implemented in a much less optimal fashion than what is assumed below. The implementation of such a scheme would require a very complex analysis of whether a point is reused within the caches overall capacity. In practice much simpler approaches, such as the least recently used replacement strategy are used. These are enhanced or restricted with additional features like a k-way nature or complex proprietary sharing schemes as for the shared L3 cache on the reference platform. Safety factors are therefore required to compensate for this uncertainty.

The memory slice size and the spatial blocking in x direction (B_x) are chosen equal and in such a way that the line criterion [124] can be served from the L2 cache. A suitable block size can be calculated from the L1 and L2 cache sizes (S_{L1}, S_{L2}), the number of unknowns U, the stencil width W and the number of intermediate buffers I:

$$\begin{aligned}(I+W)\cdot U\cdot B_x\cdot 8\,\mathrm{B/QWORD} &< S_{L1}+S_{L2}\\ (1+5)\cdot 4\cdot B_x\cdot 8\mathrm{B} &< 288\,\mathrm{KiB}\\ B_x &< 1536.\end{aligned} \tag{4.17}$$

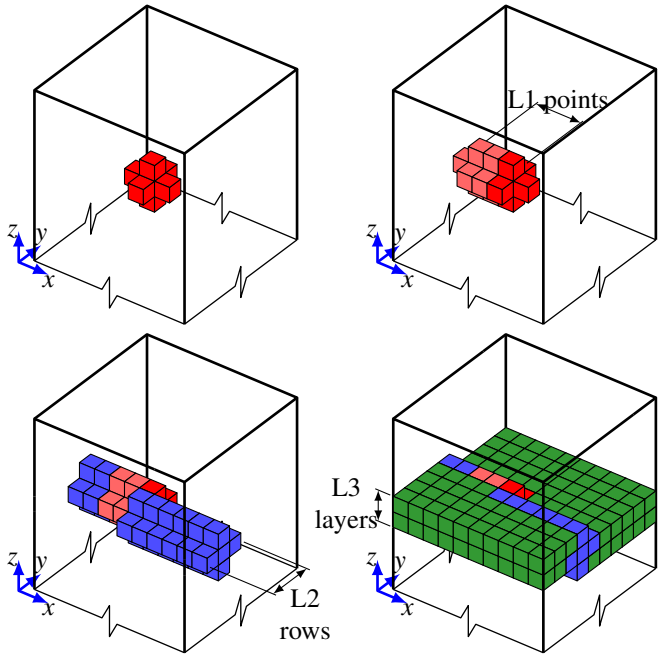
Due to the vectorization and cache line size the block size should be divisible by eight. Because of the cache associativity (see Table 4.1) it is best to avoid multiples of 512, which can cause self interference [92] requiring additional padding techniques [119]. Additionally, some space should be left available for ghost values and further overhead, especially as the L1 cache is not fully exclusive [65]. This results in the choice of $B_x = 1328$. The exact safety margin that maximizes the loop length without causing superfluous transfers must be determined empirically and the suitability of the 16% chosen here is examined later. While it would seem prudent to use only the L1 cache by choosing a loop size < 170, the resulting short loops are detrimental to in-L1 performance. The second (y) blocking direction B_y is chosen to fit into the next higher cache, i.e. L3 cache S_{L3}, according to the layer criterion. Since the L3 cache is shared between cores on the same socket only 2.5MiB are available per core. This leads to

$$\begin{aligned}\big(\big(I+(W-1)\cdot B_y\big)\cdot B_x+1\big)\cdot U\cdot 8\,\mathrm{B/QWORD} &< S_{L3}\\ \big(\big(1+(5-1)\cdot B_y\big)\cdot 1328+1\big)\cdot 4\cdot 8\mathrm{B} &< 2560\,\mathrm{KiB}\\ B_y &< 15.2\end{aligned} \tag{4.18}$$

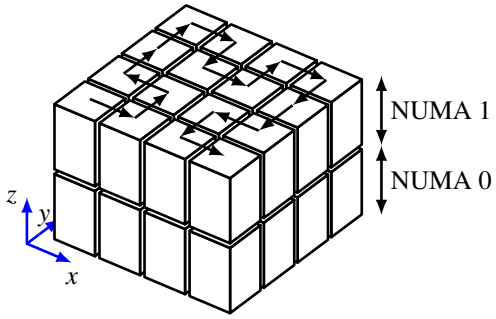
for the block size. As this includes ghost points required by the stencil and since some room should be left for overhead, the final block size is chosen to be 10. This choice may appear small, considering the stencil span of five points and the ensuing boundary overhead, but extensive tuning presented later lead to even smaller blocks as the optimal choice. The z direction is only blocked in case of multiple NUMA domains. The hardware used here is a dual socket machine with COD mode enabled and therefore acts as having four NUMA domains. Hence the domain is split into four blocks in z direction and each of these blocks is individually distributed among the seven cores belonging to each domain.

The result of these spatial blocking efforts appear rather meager when looking at the speedup VI in Figure 4.8, as it is only 7.8% faster than the plain vectorized version. However, the

coordinate splitting optimization is not included in that performance figure, which means the register spills are again severely limiting the performance. Combining both techniques, as done for VII, a 77.7% increase in throughput can be obtained. This is significantly more than the separate contributions of 7.8% and 38% and shows the importance of the interactions between different strategies. However, even with the achieved high increase in performance, there are still issues holding back the methods full potential. These challenges are addressed later.



(a) Cache correlation for a single block with ideal cache behavior and ideal configuration. The stencil's reach in neighboring blocks is not shown.



(b) Partitioning of the spatial blocks into NUMA domains and per NUMA domain along a Hilbert curve.

Figure 4.10 Spatial blocking shown for a simplified stencil and a single block along with the ensuing cache use, as well as the block partitioning.

The LBM code used in this work also supports blocking, but in a very differently manner than above. This is mostly due to the fact that there can be no reuse of cached values within a time step, as every density distribution function is accessed only once per step. Gains can therefore only be achieved through better data locality. The indirect addressing enables the blocking to be applied in a very simple manner through rearranging the address order. This causes the data in memory to be grouped by blocks and a direct loop over the indices results in a blocked access pattern to memory as well. This also makes subdivision between threads trivial. The overall SoA layout is of course retained and only the order within each field belonging to a distribution function is changed. This increases data locality which is generally favorable for caching and it has the effect of increasing the chances that accesses to neighbor cells are handled locally. These effects lead to an increase in performance of only 5% as the most influential effect of data reuse does not apply here. It is denoted as VIII in Figure 4.8 without the use of the semi-stencil and as IX in combination with the semi-stencil which provides a speedup of 22%.

An alternative form of blocking is the temporal blocking [124, 138]. It interleaves multiple time steps and revisit regions to process the next time step before all of the domain has completed

the previous step. The short time span between uses of the same point can enable an increased cache use. However, this technique is not used here as it becomes very difficult to handle for complex stencils and complex time integration. The exponential increase in data dependence for every additional time level makes it impractical in conjunction with the presented stencil. A further problem is the handling of startup and shutdown as well as the edges for such a scheme and which makes it very difficult to integrate with other optimizations and leads to very specific and rigid code. Furthermore, the eventual gains are very small as the wide data dependence of the stencils allow only very small numbers of points to remain in the limited cache at once.

4.3.6 Support for ccNUMA

This tuning step is only relevant for machines with multiple NUMA domains, hence most single socket machines will not profit from this. As multi socket systems are very common for server and cluster platforms, such as the reference hardware used here, it is an important step nonetheless. The concept of NUMA domains was introduced in Section 4.1.1.

The performance penalty from suboptimal memory usage can be quite significant. When the memory distribution is not carefully considered it can cause the cores to require large amount of data from remote memory. This is inefficient in two ways. Firstly, the latency increases significantly as the request must be forwarded to the remote socket, which then checks the caches states and, when those come back empty, requests the data from the memory controller. The data is then transferred back to the requesting socket and propagated through the caches to the core. Such long waiting times can usually not be hidden by the instruction pipeline and prefetcher resulting in stalls. The second problem with remote memory accesses lies in the limited bandwidth of the inter socket link which can easily saturate when all cores request data across this link at once. These effects become even more severe when not only remote memory is accessed, but cache lines are actively shared across NUMA domains. Every access, whether read or write, has to be synchronized between all involved cores. This requires cache lines to be evicted from one core and loaded in the other every time and even if the data has not been modified, the cache line state needs to be queried. Especially buffers as used for the semi-stencil are at risk due to the small size and use on every core. In order to ensure mostly local access to memory it is important to understand how memory is allocated by the Linux kernel. When new memory space is requested the kernel generally does not provide physical memory right away. The actual memory is only reserved on first touch (unless it was prefaulted) and is provided one memory page at a time. This strategy saves memory when not all requested memory is actually used and allows the kernel to overcommit memory. The decision in which NUMA domain the memory page is placed is decided by which core touches the memory first, unless a domain is full already. A good memory distribution can therefore usually be achieved by initializing the memory with a loop and processor distribution identical to the one for the actual computation. When the per processor memory distribution is very sparse or when huge pages are involved this may not be enough. As an example the block size used for the spatial blocking of the FDM is 1328 points. This is larger then the 4 KiB a regular memory page can hold and which results in a fine granularity when distributing blocks between NUMA domains. Only few pages are shared between domains, as can be seen from the upper diagram in Figure 4.11. It shows by color which core accesses which physical memory range. In case of regular pages the colors per page mostly corresponds to cores belonging to their respective domains. Only small portions of cores

from other domains can be seen at the top and bottom of the memory slice usage. The image completely changes when huge pages are used, as shown in the middle of Figure 4.11. Each of the four ranges of memory belonging to a NUMA domain show almost evenly distributed colors, hence cores from all domains. This is the default mode in which the reference platform is configured and it is obvious that the distribution will lead to much cross domain traffic. The reason for this lies in the huge page size of 2 MiB, which can hold about 50 AoSoA memory chunks or 65536 points. This makes it very likely for the memory blocks on a single huge page to be owned by multiple cores and the first of which to touch it has local access. This effect is even more pronounced due to the block distribution along a Hilbert curve which makes it highly unlikely for a huge page to be used only by cores in a single NUMA domain. Fortunately the domain can be decomposed in a NUMA friendly manner. For this the memory is distributed among the NUMA domain in a linear manner with regard to the virtual memory address. This can easily be achieved by dividing the entire domain in largest stride (z-direction) in the number of NUMA domains as illustrated in Figure 4.10b. Each of these blocks is then distributed among the cores of a single NUMA domain in the previously introduced spatial blocking fashion. The result is the nearly perfect access pattern shown in the bottom diagram of Figure 4.11. Using

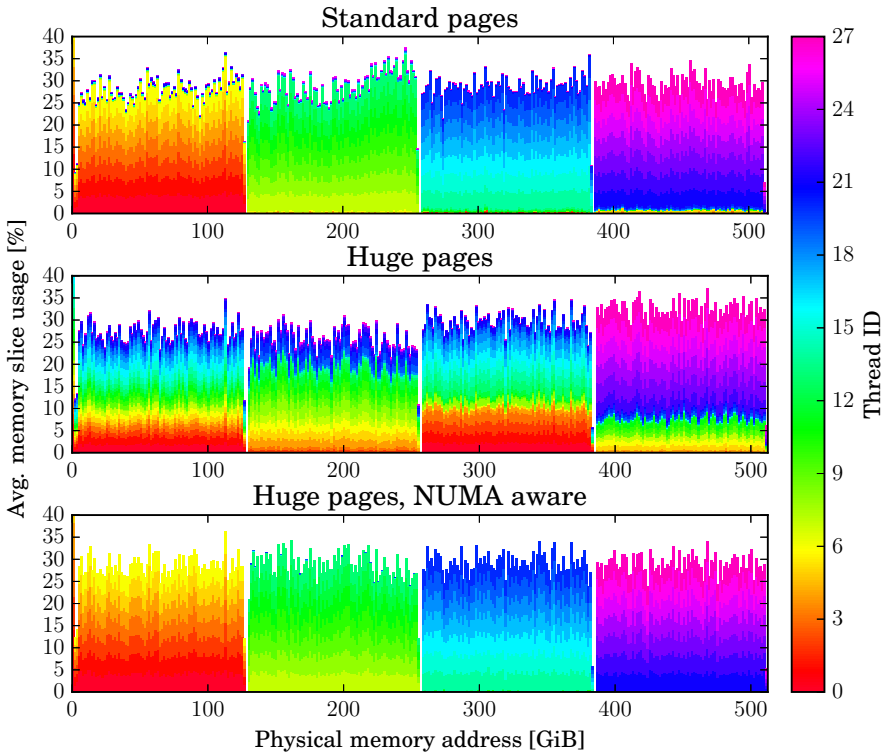


Figure 4.11 Physical memory location of memory accessed by threads/procs.

the new distribution scheme the spatial blocking VI and combined coordinate split and spatial blocking VII are tested again. Using only the spatial blocking the performance was increased by 46.2% relative to the non-NUMA aware version and is shown as VIII. For the combined approach the performance was increased by 39% and is given as IX in Figure 4.8.

The LBM's SoA memory layout is much less susceptible to these difficulties and the indirect addressing allows the memory to be allocated as a continuous block per thread independent of the domain section it represents.

4.3.7 Edge based evaluation

A further optimization technique has its origins in the high evaluation cost of divisions. Opposed to almost all other arithmetic operations, which have a throughput of 1 cycle for vectorized operations on all recent general purpose Intel CPUs, the divisions are significantly more costly. On the Sandy Bridge architecture, for which the FD code was originally tuned, the vectorized division costs 44 times as much as multiplications [33, 69]. As there are six divisions in the evaluation of the AV stabilization a significant time is spent dividing and it can stall the entire pipeline when all other independent operations have been processed. There is a way to reduce the number of divisions that is possible as the stencil is designed to have symmetrical contributions along any edge. The term containing the division is therefore identical for either side. Instead of evaluating all edges surrounding a point and summing up their contributions, the edge can be evaluated and its contribution be distributed to the adjacent points. This effectively halves the number of divisions. Of course there is some overhead as other terms cannot be split as nicely, but as the divisions are very costly quite a few operations can be compensated. Problematic is that a straight forward implementation of an edge based evaluation increases the number of memory transfers massively as per edge all adjacent points have to be loaded, doubling the memory transfers per coordinate direction. Another major drawback of moving to an edge based evaluation is the complexity involved with the required lead-in and -out to transition between point and edge based. This is especially true when this is done for the differences in all coordinate directions and with spatial blocking. As a compromise only the edges in x-direction are evaluated and the y and z-direction remain in classical pointwise processing. This has the great advantage of the caching being inherently handled correctly and the transition is only performed in a single direction. The result is the speedup of 130.2% over the vectorized version and is depicted as X in Figure 4.8. It is therefore slower than the version without the edge based evaluation which is 146.9% faster than the vectorized code. This is the result of architectural changes from Sandy Bridge to Broadwell which decreased the duration for vectorized divisions from 44 cyc to 16 cyc [33, 69]. This makes the divisions significantly faster and the additional effort for edge based evaluation outweighs the savings. It is left in the tuning sequence to demonstrate the influence of architectural changes later on.

For the LBM code there is no equivalent optimization that could be applied since there is no single very expensive operation and furthermore the 18 edges would make it very difficult to implement.

4.3.8 Nontemporal stores

A further optimization strategy is the use of non-temporal stores, which also go by the name of streaming stores. It uses a special kind of store operation available on some systems that allows the stored data to bypass the cache hierarchy. This can be useful in cases where the stored data will not be used again in the near future, as it avoids evicting other useful cached data. A second and in this case even more important feature is that read for ownership can be avoided. This

eliminates propagating the cache line with the target address to the L1 cache before it is written by the core. This can reduce the pressure on the main memory interface as the number read operations are reduced.

The LBM code is well suited for this optimization. Per update it reads the 19 incoming density distributions and writes the 19 outgoing ones without the need for the old outgoing values. This allows the total number of values transferred per cell via the main memory interface to be reduced from 57 to 38. As the LBM code is severely bandwidth limited this translates to a direct performance increase of 54.9% from V to XI. However, these results are not entirely comparable as the Intel compiler was used for XI since the non-temporal stores cannot be generated by GCC. The alternative to using a different compiler is programming in intrinsics, but this makes the code very inflexible and closely tied to a specific hardware.

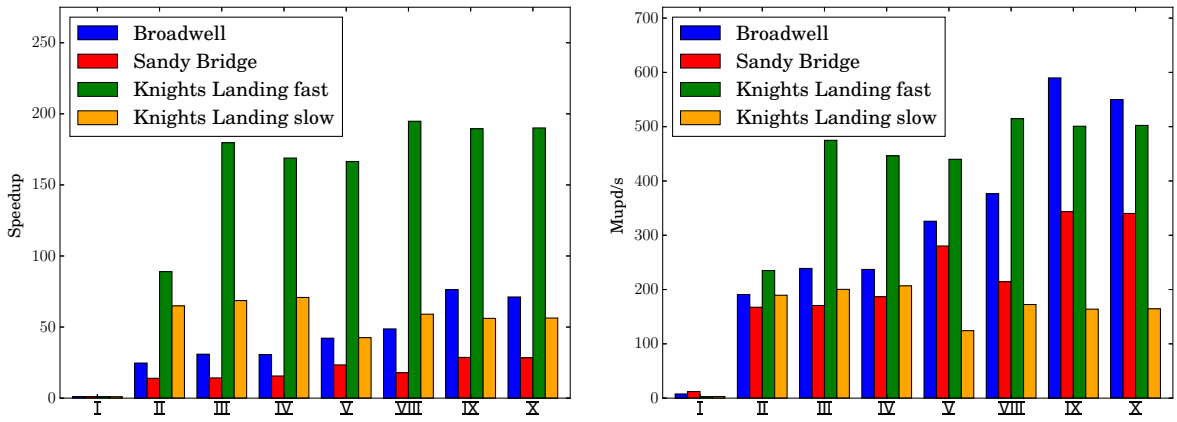
Unfortunately this technique is not applicable to the FDM. The old value is required in the calculation as it is updated with an increment. Hence the read must be performed anyways and the number of transfers would not be reduced.

4.3.9 Transfer to other platforms

There is considerable effort involved in optimizing and the previous steps are tailored to a specific platform. In order to give some insight into how well these techniques can be transferred to other platforms the steps outlined above have also been tested on two other platforms. Two alternative platforms are used. One is an Intel Xeon E5-2670 dual socket machine with 16 cores based on the older Sandy Bridge architecture. The Sandy Bridge system is of a very similar architecture to the Broadwell platform used as the reference. The optimizations are directly transferable and deviations reflect the development of CPU architectures. The other system is an Intel Xeon Phi 7210 with 64 physical cores of the Knights Landing (KNL) architecture. It is a different design altogether and uses the Intel Many Integrated Core architecture, which deviates strongly from traditional general purpose CPUs and it showcases how well the tuning steps transfer to very different platforms.

The various tuning steps shown in Figure 4.8 are given for the alternate hardware in Figure 4.12. It is important to note that the non-NUMA aware techniques (VI,VII) have been left out. As the FDM code showcases the most optimization steps it is used exclusively and hence the non-temporal store technique (XI) is forgone as well.

The Sandy Bridge system used for the analysis was not specifically prepared for benchmarking. This leads to additional differences to the Broadwell platform. The Intel Turbo Boost feature for example enables the CPU to run the single core version I at an increased clock frequency of 3.3 GHz compared to the base frequency of 2.6 GHz. This leads to the older Sandy Bridge machine outperforming the Broadwell system on single cores as shown in Figure 4.12b, but it also causes smaller speedups in Figure 4.12a. The lower speedup can also partially be attributed to the lower core count of 16. The Hyper Threading technology is disabled on both systems. The performance speedup of 14 is very good, especially considering the influence of the turbo feature. However, the same cautions as described in Section 4.3.1 apply. As opposed to the reference platform there is a small direct performance gain of 9.5% from the AoSoA memory layout IV as the TLB are more restricted and have a higher benefit from the reduce number of memory streams. While the semi-stencil V has a similar impact as on the Broadwell platform, this is not the case for spatial blocking VIII. The throughput is only increased by 15% despite



(a) Speedup **(b)** Mupd/s

Figure 4.12 Speedups and absolute update rate of the FDM code for the different tuning stages on multiple platforms: Ⅰ serial, Ⅱ parallel, Ⅲ vectorized, Ⅳ AoSoA memory layout, Ⅴ semi-stencil, Ⅷ NUMA aware spatial blocking, Ⅸ NUMA aware semi-stencil and spatial blocking, Ⅹ edge based evaluation. Adapted from [157].

identical cache sizes to the reference CPU and therefore optimal block sizes. However, the cache bandwidths are lower and the reduced core count can lead to smaller gains. The combined case of semi-stencil and spatial blocking Ⅸ exhibits the expected performance increase. The edge based evaluation Ⅹ was originally developed for the Sandy Bridge architecture due to its high cost of divisions. On a system with few cores the impact is significant, but on the server platform used here the bandwidth limitation is severe enough to cover up any increased computational throughput.

The Knights Landing platform is a completely different design from the ones described above. In Figure 4.12 two different configurations of this system are shown. This is due to an additional special random access memory (RAM) with a higher bandwidth, but limited to 16 GB. This multi-channel dynamic RAM (MCDRAM) is used exclusively in the “KNL fast” measurements, while the regular main memory is used in the “KNL slow” case. Naturally the use of the MCDRAM is preferred due to the very wide bandwidth, however the size restrictions significantly limits its use for practical applications. It is possible to integrate it into the use of regular memory, but this would require different optimization techniques that are not considered here. The system provides 64 physical cores which can process 256 threads at the same time. At 37.8% of the single core performance of the Broadwell system, each core of the KNL is significantly slower. The use of multiple cores with the regular main memory the performance increases by a factor of 65, which almost matches the number of physical cores. However, the increase by a factor of 89 using the MCDRAM shows that the code is not actually compute bound. Also the vectorization Ⅲ does not have any influence on the version with regular RAM, which additionally indicates a bandwidth limitation. The MCDRAM version with a significantly wider memory bandwidth can double the performance. Even though this does not match the theoretical capabilities of eight fold vectorization, it shows the importance of vectorization on KNL. The AoSoA memory layout and the coordinate splitting have hardly any influence. The coordinate splitting is especially important for the other platforms, on KNL however it is not required. This can largely be attributed to the doubled number of register that significantly reduces register spill. The coordinate splitting

therefore only adds overhead. One of the techniques that shows benefits also on the KNL platform is the spatial blocking. Due to the very different cache hierarchy the results in Figure 4.12 are mixed. The slight performance increase with the MCDRAM indicates that there is potential for this technique when configured suitably.

In conclusion the techniques described previously offered mixed results when ported to other platforms. While some of the steps can be applied identical to the tested architectures, others require some adjustment in the configuration or are not suitable at all. Each of the steps must therefore be considered individually for every platform and adjusted as necessary. This makes it even more important to stay flexible in the coding and tuning techniques.

4.4 Optimization and performance model interaction

As described before the tuning process is iterative with modeling the performance and applying tuning techniques. For the sake of brevity both components have been introduced separately with several performance models in Section 4.2 and a sequence of optimizations in Section 4.3. In order to show some of this interaction the following Section 4.4.1 is dedicated to the application of performance models to the fully tuned versions of the FDM and LBM code. As the models exhibit limited accuracy for these codes, the origins of these discrepancies are investigated in Section 4.4.2 alongside the influence of optimizations on the performance model accuracy.

4.4.1 Applied performance models

4.4.1.1 Roofline

The roofline model has been introduced in Section 4.2.1 for a simple implementation of the FD loop kernel. However, even for the unoptimized implementation the model showed poor accuracy in the performance prediction and it is unlikely to improve for a highly tuned version. Nonetheless these drawbacks must be weighed up against the ease of application. This is especially important in the optimized context as that generally makes modeling more difficult.

The hardware on which the tuned code is executed remains the same as for the simple version. Additionally the measure were chosen to maximize the independence of the machine balance from the code. Hence the machine balance in terms of instructions and quadwords (or doubles) is still

$$B_m = \frac{153.6}{28 \cdot 2 \cdot 2.6} \frac{\text{GB/s}}{\text{GINSTR/s}} = 0.132 \frac{\text{QWORD}}{\text{INSTR}}. \tag{4.19}$$

This is of course the result of the total main memory bandwidth and the sum of the peak computational throughput of all cores. A detailed hardware specification can be found in Table 4.1.

The code balance B_c on the other hand differs from the one given in equation (4.5) as the operation count of the loop kernel and the bandwidth requirements have been modified in the optimization process. In fact the loop kernel has been split into four consecutive loops processing the differences in x-, y-, and z-direction separately and updating the RHS afterwards. This is also reflected in the instruction count in Table 4.5, which provides each loop separately. The loop in x-direction is further subdivided into the peel, main, and remainder part. This subdivision of the x-direction loop is required for effective vectorization as the two outermost points on either end of each memory chunk have to access points in neighboring blocks at non-constant

offsets. Since loops with conditionals can only be vectorized by the compiler in very specific circumstances these critical points were moved outside the main loop. As a consequence the total number of instructions and cycles given in Table 4.5 depend on the memory chunk size. The additional expense for the leadin and -out at the block edges are distributed among the regular evaluations. Also note the different unroll factors per sub-loop, which allows the compiler to carry out inter-iteration optimizations and reduces the cost of evaluating the exit condition. The peel and remainder loop are not unrolled as their iteration count is only two. The main loops however are unrolled by a factor of four, matching the vectorization. The update loop is also vectorized and unrolled by an additional factor of five, leading to a total unroll factor of 20. As the memory chunk size of 1328 is not evenly divisible by this factor the compiler automatically generates additional peel and remainder loops. These are also vectorized and the cost difference to the unrolled loop is marginal. They are therefore not treated separately. Additionally these peel and remainder loops require the same number of floating point instructions as the regular loop and therefore do not impact the theoretical roofline model at all. The instruction count in

loop section	unroll factor	add/ sub	mul	div	fma/s	min/ max	IACA analysis [cyc]
x-dir peel	1	30	16	2	27	6	46.55
x-dir main	4	30	16	2	27	6	49.5
x-dir remainder	1	30	16	2	27	6	64
y-dir	4	32	14	2	26	6	51
z-dir	4	32	14	2	26	6	50.95
update RHS	20	0	20	0	20	0	30.5
full update	4	94.27	48.14	6.02	83.24	18.05	158.07

Table 4.5 Floating point instruction count for processing the split loops of the fully optimized loop kernel alongside the cycles estimated by Intel IACA. The total is calculated for a block size of 1328.

Table 4.5 does not take into account account the differing execution cost of the divisions. As the vectorized division takes 16 times as long as the other instructions [33, 69] the cost per four updates is equivalent to 96.32 instructions at a throughput of 1 cyc.

Unlike the number of instructions, the required bandwidth can be determined very easily for the optimized code. Under the assumption that all implemented caching techniques work as intended, only the unknowns for a single point must be loaded from main memory per update. Combined with reading and writing the RHS, the required number of fields is 3 with 4 unknowns per point and an unroll factor of 4 or equivalently 48 QWORD or 384 B. For details on the implemented caching strategy see Section 4.3.5. With the number of operations and the data requirements established the code balance is calculated as

$$B_c = \frac{48}{340.0} \frac{\text{QWORD}}{\text{INSTR}} = 0.141 \frac{\text{QWORD}}{\text{INSTR}}. \tag{4.20}$$

The machine and code balance ratio then follows to be

$$\frac{B_m}{B_c} = 0.934. \tag{4.21}$$

With reference to equation (4.2) this indicates rather balanced code requirements and hardware capabilities, but with a slight tendency towards bandwidth limitation. The better balance between code and hardware compared to the original loop kernel with a balance of 0.251 is the result of the tuning, which of course tries to optimize the hardware usage. Naturally the overall performance is at a higher level with 1600 MUPD/s at the bandwidth limit. This is much higher than the measured 589.9 MUPD/s and shows that there are large discrepancies and which are addressed by the refined models in later sections.

The roofline model for the LBM code is applied in a similar fashion as for the FDM version and since it is run on the same platform the machine balance B_m remains identical. Naturally the code balances differ, but the operation count is much easier to determine. Because of the SoA memory layout there are no memory chunk sizes to consider and no lead in and out to account for. However, the indirect addressing required for the pull operation requires additional loading of 18 addresses. The address of the central (or zero) density distribution function can be determined directly and does not have be looked up. The remaining 18 addresses are of 64 bit size to enable addressing of enough memory for the large test cases presented later. The loop kernel for the LBM is split in even more sub loops than the FDM kernel. As they are all unrolled by the same factor, their contributions, given in Table 4.6, can be summed up directly. Furthermore the collision operator does not require divisions, which further simplifies matters as all instructions execute in a single cycle. The bandwidth requirements can easily be calculated from the 18

loop section	unroll factor	add/ sub	mul	div	fma/s	min/ max	IACA analysis [cyc]
pull	4	45	2	0	4	0	173.95
NE SW	4	3	1	0	7	0	6.6
SE NW	4	3	1	0	7	0	6.6
TE BW	4	3	1	0	7	0	6.6
BE TW	4	3	1	0	7	0	6.6
TN BS	4	3	1	0	7	0	6.6
BN TS	4	3	1	0	7	0	6.6
N S	4	2	1	0	7	0	6.0
E W	4	2	1	0	7	0	6.0
T B	4	2	1	0	7	0	6.0
full update	4	69	11	0	67	0	231.55

Table 4.6 LBM Floating point instruction count for processing the split loops of the fully optimized loop kernel alongside the cycles estimated by Intel IACA.

addresses and 19 density distribution functions that are read for the collision operation, as well as further 19 density distribution functions that are written post collision. The total number of values transferred via the main memory interface per update is 56 or 448 B. Due to the nontemporal stores employed in the fully tuned code the RFO is omitted, which would otherwise add a further 152 B. As vectorized instruction are employed the data transfer requirements must further be

# of cores	1	2	4	8	16	28
memory BW [GB/s]	12.1	27.6	52.2	85.6	108.6	111.6
ratio of single core BW	-	2.3	4.3	7.1	9.0	9.2

Table 4.7 Measured sustained memory bandwidth as established with the STREAM (add) benchmark and scattered thread distribution.

multiplied by a factor of four. The resulting code balance is

$$B_c = \frac{224}{147}\frac{\text{QWORD}}{\text{INSTR}} = 1.524\frac{\text{QWORD}}{\text{INSTR}}, \tag{4.22}$$

which results in a machine to code balance ratio of

$$\frac{B_m}{B_c} = 0.087. \tag{4.23}$$

This is clearly < 1 indicating that the code on the specified platform is operating at the bandwidth limit. The theoretical throughput at this limit is 342.857 MUPD/s. Without the RFO tuning the update rate would drop to 256.0 MUPD/s, underlining its importance in the bandwidth limited case. Running the LBM code on a single core the code is estimated to be at the compute limit and generates a throughput of 141.50 MUPD/s. This means that memory bandwidth saturation will occur with three or more cores. Either way the measured update rate of 217.89 MUPD/s is significantly lower, a discrepancy analog to that of the roofline predictions for the FDM loop kernel. Hence, more detailed models are applied in the following.

4.4.1.2 Enhanced roofline

The enhancements to the basic roofline model introduced in Section 4.2.2 can also be applied to the fully tuned FDM and LBM loop kernels. The most important difference is the use of the sustained memory bandwidth that has been established by measuring the ADD case of the STREAM benchmark suite [72]. The measured bandwidth for different number of cores is depicted in Figure 4.6a and a selection is also given in Table 4.7. The 28 core sustained bandwidth is used for the enhanced machine balance and leads to $B_m = 0.192$ as previously introduced in equation (4.12).

The enhancements to the code balance of the FDM code are considered first. The IACA analysis replaces the manual estimate of the arithmetic complexity. The IACA results are provided in Table 4.5, separated into the component loops. The total is again computed with respect to the memory chunk size. Note that the number of cycles produced by the IACA analysis are for the number of updates given by the unroll factor, a total of four for the full update. The bandwidth requirements in the code balance do not change compared to the ones used for the theoretical roofline model. They again correspond to four updates since the instructions are taken as vectorized. The code balance is then

$$B_c = \frac{4 \cdot 12}{158.1}\frac{\text{QWORD}}{\text{cyc}} = 0.304\frac{\text{QWORD}}{\text{cyc}} \tag{4.24}$$

and the machine to code balance ratio is

$$\frac{B_m}{B_c} = 0.631. \tag{4.25}$$

This shows a stronger tendency towards memory bandwidth limitation than the purely theoretical roofline model. The bandwidth limited update rate is 1163 MUPD/s, which is still 97% higher than the actual measurements. Even though this is a significant improvement in accuracy, there is much room for further improved by the ECM model, which also takes caching effects into account. These are a very important feature in the case of the fully tuned kernel as it makes heavy use of the cache hierarchy.

For the LBM kernel the machine balance stays the same as well and the code balance is also established with the help of Intel IACA. The partial contributions of the split loops are given in Table 4.6. It also provides the total in number of cycles required for updating four points. This leads to a code balance and a balance ratio of

$$B_c = \frac{4 \cdot 56}{231.6} \frac{\text{QWORD}}{\text{cyc}} = 0.967 \frac{\text{QWORD}}{\text{cyc}} \qquad \frac{B_m}{B_c} = 0.199. \tag{4.26}$$

Just as for the roofline model based on theoretical figures this shows a clear bandwidth limitation and therefore a maximum throughput of 249 MUPD/s. The single core, compute limited throughput is predicted at 44.9 MUPD/s and the bandwidth saturation point is estimated to be reached with six or more cores.

4.4.1.3 Execution-cache-memory

The next refinement step with regards to performance models is the ECM model. It was already introduced in Section 4.2.3 and it augments the measurement enhanced roofline model by additionally taking the cache hierarchy into account. The cache usage in the basic example from the introduction was designed to be trivial. For the fully tuned code however, the caches are an essential feature of the tuning and are used extensively. This makes the ECM well suited as a performance model, but it also makes determining the cache usage very complex. Fortunately the block sizes have been tuned to a particular cache usage pattern in the optimization process. This is hardware architecture specific, but as the same platform is used for tuning and for modeling the configuration matches. Details of the cache architecture on the reference platform can be found in Table 4.1 and on the optimization of the cache utilization in Section 4.3.5.

Under the assumption that the tuning results work as intended, the L1 cache is mostly occupied with the data from register spills, the stack, and factors used in every loop iteration. Also served from the L1 cache are all but the first point per update in the loop over the differences in x-direction. The L2 cache is significantly larger and is able to cache data at a decent loop length. The memory chunk size or block size in x-direction has been adjusted to make optimal use of the L2 cache, 1328 points in this case. This means that the first point for the differences in x-direction and all but the first of the difference in y-direction are provided by the L2 cache. This corresponds to four rows of points with the length of the x-block size and one additional point. This row criterion [124] is also illustrated in Figure 4.10a. Additionally the buffer that holds one row of points with intermediate results is also served by the L2 cache, as well as the values for the RHS which must pass through the L2 cache for the update. This is summarized in Table 4.8

read / load [4CL]				
kernel part	MM-L3	L3-L2	L2-L1	L1-core [B]
x		U	U+B	3728
y		4U	5U+B	4240
z	U	4U	5U+B	4368
collect	R	R	R+B	512
write / evict / store [4CL]				
kernel part	L3-MM	L2-L3	L1-L2	core-L1 [B]
x			B	768
y			B	704
z			B	640
collect	R	R	R	256

Table 4.8 Transfers between caches for the fully optimized spatially blocked FD loop kernel. Transfer of four cache lines (= 256 Byte) of unknowns is denoted by U, four cache lines from the RHS field by R and four cache lines from the buffer by B. The peel and remainder loops are not shown. The transferred data between L1 cache and core were obtained through an IACA analysis.

loop kernel component	per hierarchy level duration [cyc]					accumulated duration [cyc]	Symbol
	IACA	Core-L1	L1-L2	L2-L3	L3-MM		
x peel	93.1	16.5	3	2	0	93.1	$T_{x\text{-}peel}$
x	99	58.25	12	8	0	99	T_x
x remainder	128	37	3	2	0	128	$T_{x\text{-}remainder}$
y	102	66.25	28	32	0	126.25	T_y
z	101.9	68.25	28	32	31.7	159.95	T_z
collect	12.2	8	12	16	63.5	99.5	$T_{collect}$

Table 4.9 Transfer duration in cycles between cache levels for the fully optimized spatially blocked FD loop kernel per update of one cache line.

in the L2 to L1 cache transfer column, where "U" are the four unknowns per point with one cache line each, "B" the four values per buffered point also in cache lines and "R" four values of the RHS in cache lines being updated. Just as the L2 cache holds rows of the block being processed, the much larger L3 cache holds layers of these blocks. The block size in y-direction is chosen in such a manner that four layers of points can reside in each core's share of the L3 cache. As a result only the first point in z-direction and the RHS has to be loaded from main memory for every update. This is also reflected by the MM to L3 column in Table 4.8.

With the number of transfers between the caches known, these can be converted into the transfer durations that are ultimately required for the ECM model. The transfer rates between cache levels, main memory, and core have been established in Section 4.2.3 and are summarized in Table 4.3. Here, they are used analog, but for each loop separately. This leads to the timings given in Table 4.9. The per level timings are then combined according to equation (4.15). The results of this step are listed in Table 4.9 separated between the loops.

When combining the separate per loop timings the block size in x-direction B_x must be taken into account. The cost for the peel and remainder loop is distributed among all points per block, resulting in the total duration

$$T_{total} = \frac{T_{x\text{-}peel} + T_{x\text{-}remainder} + \frac{T_x}{8} \cdot (B_x - 4) + \frac{T_y + T_z + T_{collect}}{8} \cdot B_x}{B_x} \tag{4.27}$$

for updating one cache line or eight points. The $T_\square$ are the cycles required per update for the different loops given in Table 4.9. Inserting a block size of $B_x = 1328$ the total number of cycles per update is 60.7 cyc. On the reference platform this translates to 42.8 MUPD/s per core. The multi-core scaling assumptions for the ECM model have already been discussed in Section 4.2.3. The per core throughput scales linearly with the number of threads until the BW limitation of 1163 MUPD/s is reached. Here 27.2 cores would be required, which places the available 28 cores slightly into the bandwidth bound regime. This can also be seen in Figure 4.13.

Application of the ECM model to the LBM loop kernel differs only slightly from the FDM kernel. From the number of cache transfers summarized in Table 4.10 it is apparent that the loop is split into many component loops. These are almost all identical with the exception of the first “pull”-loop. This “pull”-loop reads the 19 incoming distribution functions spread out across the memory and stores them in the buffer in an ordered fashion for further use by the other loops. The memory is addressed indirectly for the pull operation apart from the central density distribution for which the address can be calculated directly. The result is an additional load of 18 addresses of 8 B. In conjunction with the SoA memory layout the 19 distribution functions are likely loaded from 19 distinct memory regions. The additional load of the memory addresses places a high load on the main memory interface and TLBs. This load is somewhat mitigated by the SoA memory layout and the linear iteration scheme which assures that all values loaded within a CL are used. Nonetheless, 19 cache lines of addresse and of data have to be transferred through the entire cache hierarchy, as shown in Table 4.10. The L1 cache to core transfers are again extracted from the Intel IACA analysis and correspond to the eight updates required for processing a cache line. Apart from the high load requirements the “pull”-loop also performs many write operations. Most of these pertain to the buffer used for the semi-stencil and in which most incoming density distribution functions are rewritten in an order matching the subsequent vectorized processing. Additionally, the local velocity and density are buffered. As the buffer is relatively small and frequently used it is unlikely to be evicted further than the L2-cache. Apart from the buffer the “pull”-loop also processes the density distribution function associated with the zero velocity. This value are directly written to the main memory with the help of non-temporal stores. The “pull”-loop loads and rearranges all data required for the remaining nine loops in a SIMD compatible layout. These loops can therefore be executed without further reads from main memory. The equilibrium values and the required incoming density distribution functions are obtained from the buffer. The buffer data is usually served from the L2 cache. The loops that update two opposing and axis aligned density distribution functions require four values from the buffer. The loops that process the distribution functions on the diagonal require an additional value. This is caused by the need for two local velocities that form the diagonal. All loops write the outgoing density distributions directly back to main memory through non-temporal stores. As the memory for outgoing distribution functions is addressed directly and accessed consecutively these operations can easily be vectorized and forego the need for address lookups.

	read/load [CL]			
kernel part	MM-L3	L3-L2	L2-L1	L1-core [Byte]
pull	19F+18I	19F+18I	19F+18I	4032
NE SW			5B	320
SE NW			5B	320
TE BW			5B	320
BE TW			5B	320
TN BS			5B	320
BN TS			5B	320
N S			4B	256
E W			4B	256
T B			4B	256
	write/evict/store [CL]			
kernel part	L3-MM	L2-L3	L1-L2	core-L1 [Byte]
pull	F		22B	1728
NE SW	2F			128
SE NW	2F			128
TE BW	2F			128
BE TW	2F			128
TN BS	2F			128
BN TS	2F			128
N S	2F			128
E W	2F			128
T B	2F			128

Table 4.10 Transfers between caches for the fully optimized LBM loop kernel, separated into its' subloops. The "F" corresponds to a load or store of a density distribution function, "I" to a load of an address for the indirect addressing and "B" to a single buffer entry. All transfers are given in cache lines, except for the L1 cache to core transfers, which are given as Byte. The peel and remainder loops are not shown, since they are essentially identical, but do not use full cache lines.

In the following a remark on write operations using non-temporal or streaming stores and how they influence the ECM model is given. In theory these stores fully bypass the caches, as the stored data is not written to the cache, but directly passed on to the main memory. The specific behavior of these store operations are not fully published for the reference platform, but several known prerequisites and measurements allow for some assumptions regarding the modeling. If the non-temporal stores write consecutive 64 B blocks the read-for-ownership is omitted and the data is directly written to main memory [65]. This is the effect for which the non-temporal stores are used here and it can be directly verified by using the built-in performance counter to measure the transferred data at every cache level. This measurement can also be used to determine whether the non-temporal stores also share the available bandwidth between caches. Unfortunately these measurements are inconclusive, as the streaming stores are counted at some cache levels on some architectures. It could of course be a matter of the exact implementation of the different performance counters. However, with these uncertainties and for the ease of use of

loop kernel component	IACA compute	per hierarchy level duration [cyc] Core-L1	L1-L2	L2-L3	L3-MM	accumulated duration [cyc]
pull	347.9	63	60	77	300.05	500.05
NE SW	13.2	5	5	0	12.99	22.99
SE NW	13.2	5	5	0	12.99	22.99
NE SW	13.2	5	5	0	12.99	22.99
SE NW	13.2	5	5	0	12.99	22.99
TE BW	13.2	5	5	0	12.99	22.99
BE TW	13.2	5	5	0	12.99	22.99
TN BS	13.2	5	5	0	12.99	22.99
BN TS	13.2	5	5	0	12.99	22.99
N S	12	4	4	0	12.99	20.99
E W	12	4	4	0	12.99	20.99
T B	12	4	4	0	12.99	20.99

Table 4.11 Transfer duration in cycles between cache levels for the fully optimized LBM kernel per update of one cache line.

the ECM model the duration for streaming stores has also been determined experimentally. This was done analog to the regular stores with the suite of STREAM microbenchmarks [72]. For the reference platform the streaming stores are determined to take 6.494 cyc for the transfer of one cache line from the core to the main memory. The per loop timings are again evaluated according to equation (4.15) and a summary can be found in Table 4.11. The total duration of updating one cache line is trivial to determine for the LBM loop kernel, since no lead-in or -out has to be considered. Simply adding the cycles results in a total of 746.9 cyc for the eight updates of a CL. On the reference platform this translates to 27.85 MUPD/s per core. As the bandwidth limit is at 249.2 MUPD/s at least 9 cores are required to fully saturate the main memory interface.

4.4.1.4 Comparison

In the following the quality of the models introduced above is assessed by comparing them to measurements of the FDM and LBM code. The models and measurements of the fully tuned FDM loop kernel are shown in Figure 4.13. The measured update rates are based on the best of ten runs for each number of threads. The domain with the shape and size of $27888 \times 200 \times 880$ points is updated four times per run. The number of threads used for the measurements is selected to be able to evenly divide the grid between the NUMA domains. The number of spatial blocks of which the full domain is composed, is designed to be evenly divisible by the number of threads per NUMA domain. This avoids imbalances in the per core execution times that are not be captured by the models.

Figure 4.13 demonstrates significant differences between the measurements and predictions, as well as between the different models. There are two major effects that should be captured by the models. The first one is the peak performance when all cores are used and the memory interface is saturated. It can be clearly seen, that estimating this performance figure purely based on theoretical memory throughput greatly overestimates the update rate . Using a memory

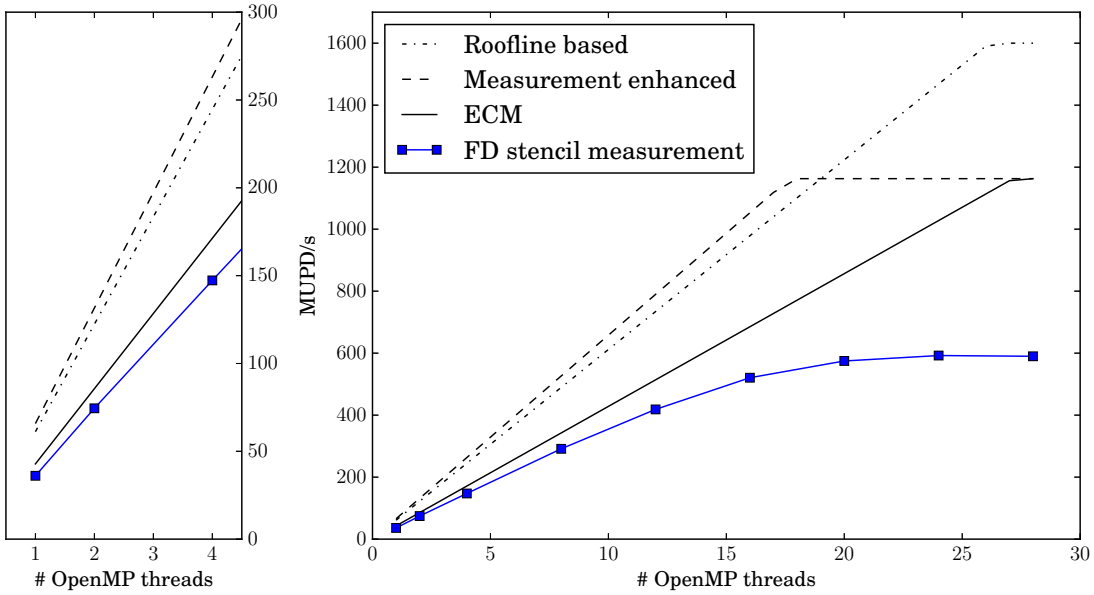


Figure 4.13 Comparison of the roofline and ECM models with measurements over the number of cores. Adapted from [157].

throughput based on measurements of a microbenchmark designed to saturate the main memory interface yields significantly better results. This sustainable memory bandwidth is used by the significantly more accurate tool assisted roofline and ECM models. However, even with the sustainable bandwidth the actually achievable throughput is still overestimated by 97%. This is also the reason why the ECM model predicts that the bandwidth limited region is barely reached. At the same time the measurements show a behavior that is very characteristic for a bandwidth limitation, even for 20 threads.

The other effect that should be recovered by the models is the scaling at lower core counts, when the execution is limited by the arithmetic throughput and the main memory interface is not yet saturated. As can be seen this is not represented adequately by either of the roofline models, even with the help of a detailed, tool assisted analysis of the loop kernel. The measured single core throughput is overestimated by 70% and 82% by the roofline models without and with the enhancements, respectively. The ECM model is much more accurate in this regard at a throughput overestimated by only 19% and which suggests that the cache hierarchy plays a crucial role. An investigation into the cause of these mismatches between the models and measurements is carried out in Section 4.4.2.

For the performance models of the LBM loop kernel the same tendencies as with the FDM kernel can be seen. The predictions of the models and the measurement of the actual kernel are compared in Figure 4.14. The measurements have again been performed with thread counts leading to a balanced used of the NUMA domains. The roofline model that is based purely theoretical values significantly overestimates the achievable performance. This is not surprising, since the same effects occur for the roofline model of the FDM and which stem to a large degree from considerable discrepancy between the sustained memory bandwidth and the theoretical bandwidth. In total this leads to an overestimate of the throughput by 57% in the bandwidth

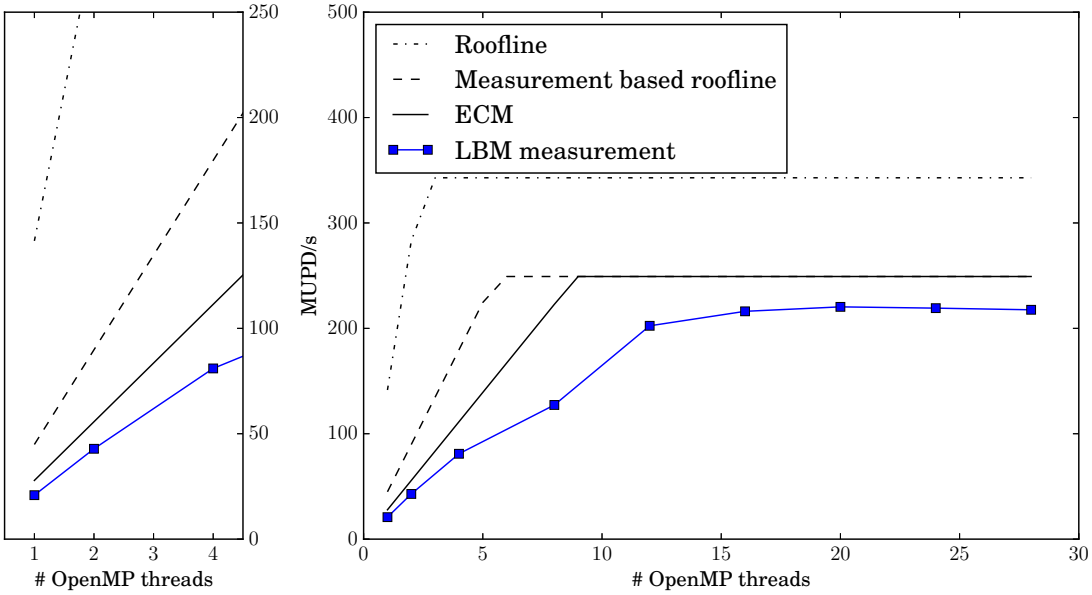


Figure 4.14 Roofline, ECM, and measurement comparison over the number of cores for the LBM code.

limited regime. The throughput in the non-bandwidth limited case is overestimated even more severely by 578%. The error in the compute limited case is in large parts caused by register spills of the loop kernel that are not accounted for by the manual instruction count. The enhanced roofline model was improved with the automated analysis of the loop kernel and does take these effects into account and substantially reduces the overestimate to a factor of 1.2. While this is a significant improvement, it is still a very large error that renders these results unusable. The use of the sustained memory bandwidth increased the accuracy of the estimate in the BW-limited regime to a relative error of only 14%, which is a very good agreement in the performance modeling context. Further improvements can be achieved with the ECM model, which reduced the relative error to the measurements for the scaling below the bandwidth limit to about a quarter at 33%. Since the estimates in the bandwidth limited case are identical to the measurement enhanced roofline model no additional improvements can be made in this regime. Of course the remaining differences also influences the quality of further estimates, such as the saturation point, which always occurs at lower core counts for the measurements. Not all of the discrepancies noted above can be traced back to their origin and some can always remain due to unpublished and unknown behavior [49]. However, many issues can be investigated and corrected for, as done in the following.

4.4.2 Optimization impact on model accuracy

As seen in the previous section some models exhibit a significant error. This is improved by the more advanced models and could most likely be further reduced by additional refinements. However, at some point the setup of the model is more expensive than simply implementing and measuring the actual implementation, which defeats any predictive capabilities the models

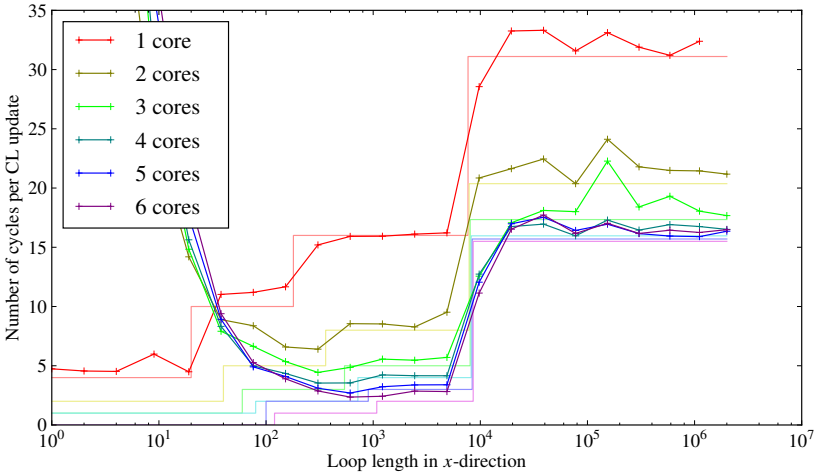
might have. In order to understand the limitations set forth by certain modeling approaches, it is important to investigate the reasons for the deviations.

In the following the ECM model is used in two different approaches to determine the source of the errors. In the bottom up approach the model is applied to a very simple loop kernel, for which the model is known to be accurate. The kernel is then successively transformed towards the full FD loop kernel to determine when the predictive capabilities deteriorate. The top down approach, as the alternative method, analyzes the ECM model for the fully optimized kernel and successively leaves out tuning techniques until the estimates and measurements match.

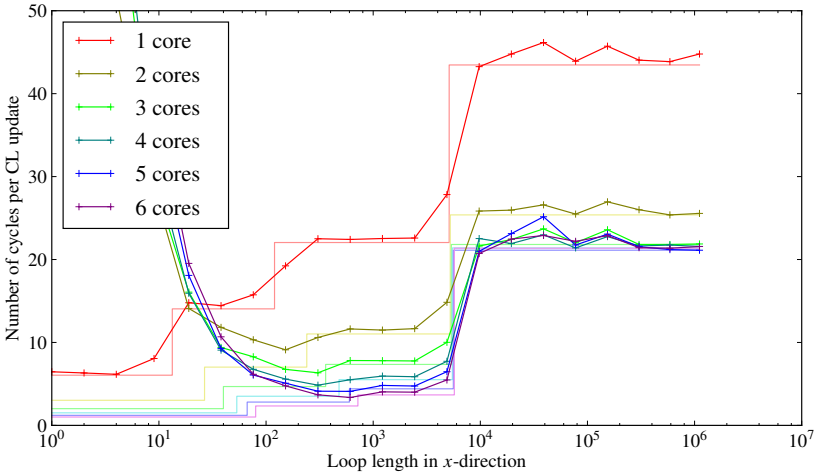
4.4.2.1 Bottom up

There are a wide range of possible origins to the discrepancies between model and measurement and, as introduced above, this bottom up approach attempts to determine the source of these difficulties by analyzing successively more complex examples. The basic loop kernel used in this work is a slightly modified “add” benchmark from the STREAM suite. This basic version is expanded in two different regards. For one thing the number of fields is increased and also a stencil like offset in the array accesses is introduced. Later the combined influence is evaluated. At every stage of the various test cases the ECM models are compared to the measurements and the increase in error is determined. Due to the number of tests and the complexity involved, the configuration of the model is automated. This way many more architectural features can be taken into account than is practical when manually configuring the model. Especially for the combined test cases that test the interaction between stencil offsets, loop and array length, the number of cores, and the cache levels the setup becomes very involved, as seen from the ECM predictions in Figure 4.17. Differing from the previous measurements these analysis are not been performed on the reference platform used for the full FDM and LBM analysis above. Instead a workstation with an Intel Sandy Bridge E5-1650 with six cores running at 3.2 GHz is used. The hardware architectures are very similar, as both CPUs are part of the same family and the same problems in performance modeling have been observed on both platforms. The performance difference is also uncritical, as only increases in the relative error between modeled and measured performance are relevant and not absolute performance. All means used to reduce the noise in performance measurements on the cluster node are also employed on the workstation, i.e. Hyper Threading and Turbo Boost are disabled and the performance governor is used in conjunction with disabled C-states.

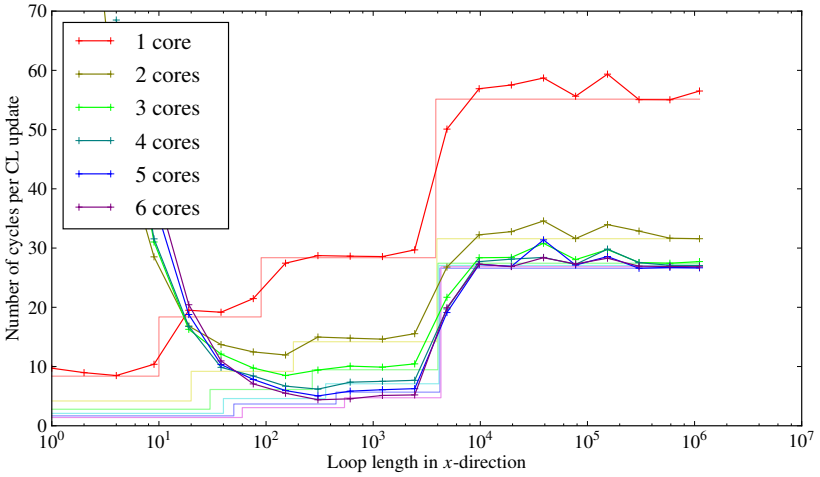
The bottom up approach starts out with an analysis of the loop kernel `A[] = A[] + B[]`, which performs the addition of two arrays in place. The model and the measurements given in Figure 4.15a show excellent agreement, as expected from earlier analysis of this operation. The deviations for very short loop length on multiple cores are due to the very short run time of each loop repetition. This causes the synchronization of the OpenMP environment to have a significant impact. The single core version is run without OpenMP and show excellent agreement also for the very short loops operating within a lower cache level. The steps in the predicted cyc/UPD originate from the different cache levels. The difference in jump size and loop lengths are due to overall cache size increasing with the number of cores. These jumps are accurately modeled by the automated model generation. The most relevant part of the diagrams in Figure 4.15 is the region with high loop iteration counts. It operates from the main memory, which matches



(a) Add two arrays, `A[]=A[]+B[]`.



(b) Add three arrays, `A[]=A[]+B[]-C[]`.



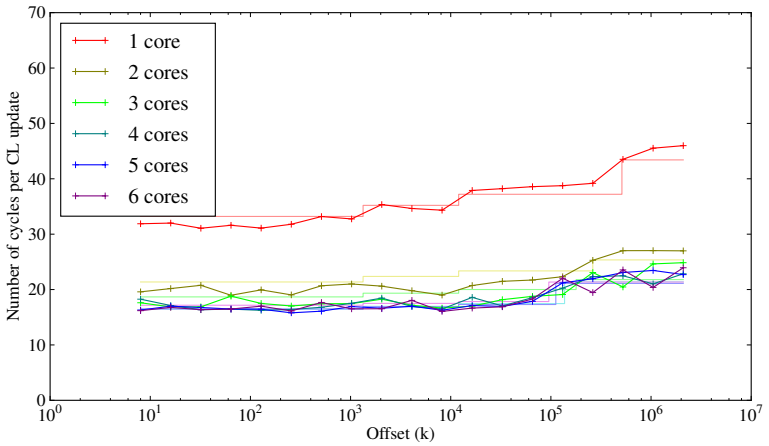
(c) Add and multiply four arrays `A[]=(A[]+B[])*(C[]-D[])`.

Figure 4.15 Variations of the STREAM "add" benchmark with increasing number of arrays.

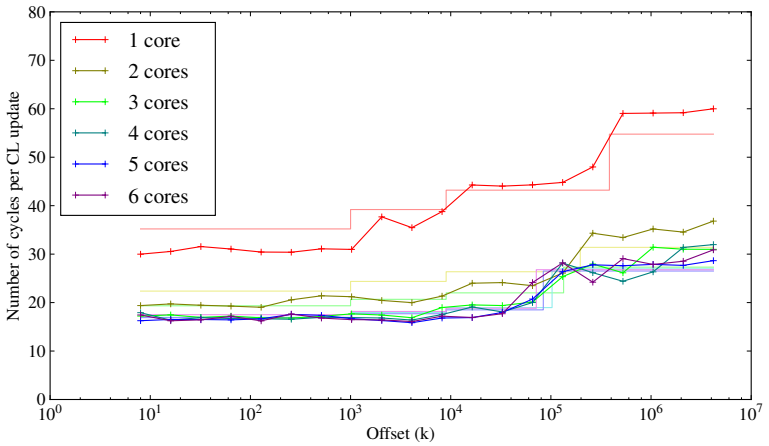
typical simulation settings with domain sizes significantly larger than the cache capacity. For all number of cores the performance in that regime is predicted well, apart from a few loop iteration counts. These specific counts induce unwanted side effects from advanced hardware features, such as cache thrashing. The increase in the number of fields is done in two steps. The performance of the version utilizing three fields `A[] = A[] + B[] - C[]` is given in Figure 4.15b and the version with four fields `A[] = (A[] + B[]) * (C[] - D[])` is given in Figure 4.15c. There are no significant changes compared to the two field version, apart from the overall increased number of cycles. The prediction quality remains steady in the cyc/UPD as well as the correlation between loop length and cache boundaries. Since the FDM code uses four fields, these tests are not expanded further and the number of fields is most likely not the cause of the large prediction errors.

The other stencil property that is tested in isolation is the access to arrays at offsets. Three different versions are used. The first one employs a single offset in `A[] = A[] + B[] - B[+k]`. The performance depending on the offset `k` are shown in Figure 4.16a. The overall loop length is kept constant for these test and is chosen large enough to ensure operation within the memory bound regime. Of course small offsets lead to caching as elements are revisited before they are evicted from cache and which is also present in the diagrams. The second version uses the same offset as the first test case, but in either direction from the center in a symmetrical fashion `A[] = (A[] + B[-k]) * (B[] - B[+k])` and again the results of the offset variation are shown in Figure 4.16b. The measurements for this case are slightly noisier and the error is also a bit higher, which indicates that the offset memory access contributes to the modeling error. For further investigation a version using two independent offsets `k0` and `k1` is used in `A[] = A[] + c*((B[-k0] + B[]) * (B[-k1] - B[]) - (B[] + B[+k0]) * (B[] - B[+k1]))`. The results in Figure 4.16c show the variation of the offset `k1` while retaining a constant loop length and offset `k0`. Again the error increases compared to the basic benchmark. However, the error is not significantly larger than for the version with one symmetric offset. It should be noted that the measurements are quite noisy in the range where both offsets are almost identical and additional cache use takes place. This effect is represented very accurately with the automated model configuration.

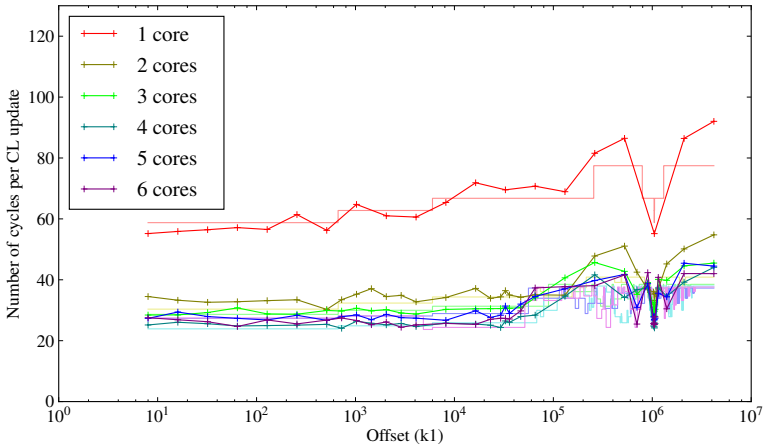
With the previous tests it was established that the use of offset array accesses have a larger impact on the modeling accuracy than the number of fields on which the loop kernel is operating. In the next test the interaction between the two techniques is investigated to determine whether the error increase may be exacerbated. Figure 4.17 presents the results for the combination of three fields with a single symmetric offset `A[] = A[] + c*((B[+k]+B[]) * (C[+k]+C[]) - (B[]+B[-k]) * (C[]+C[-k]))`. Shown is the dependence on the offset `k` for a constant and large loop length. The error between model and measurement is not significantly larger than for the single field version, despite the ECM model estimates becoming very complex for this test case. This can be seen in Figure 4.17, where large offsets `k` cause the memory accesses to interact between cores. This effect is more pronounced for larger core counts as the per core loop size and the relative distance decreases. The resulting complex interactions can only sensibly be determined through automated tools. Note also that the measurements mostly do not exhibit this fluctuating behavior. Even with a closer analysis of that specific regime the sudden decreases in cyc/UPD are only occasionally detected. This is likely due to a more complex cache behavior than assumed for the model, as well as slight timing variations between cores compared to the ideal synchronization assumed in the estimate.



(a) Single offset on domain size $19493 \times 10 \times 10$, `A[]=A[]+B[]-B[+k]`.

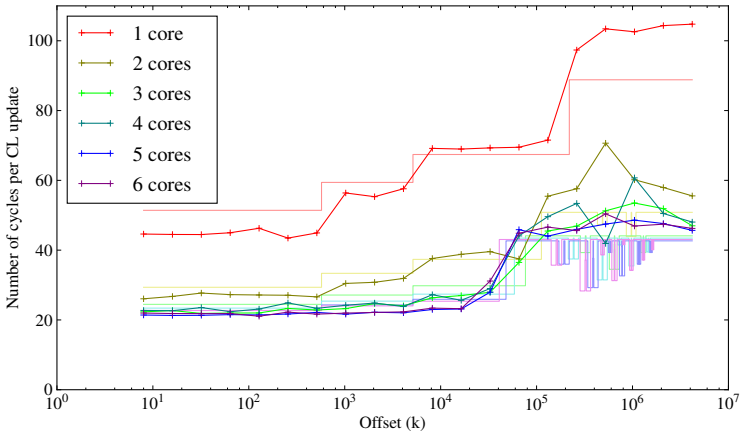


(b) Symmetric offsets on domain size $19493 \times 10 \times 10$, `A[]=(A[]+B[-k])*(B[]-B[+k])`.



(c) Two symmetric offsets on domain size $19493 \times 10 \times 10$ and with offset `k0` = 1048576, `A[]=A[]+c*((B[-k0]+B[])*(B[-k1]-B[])-(B[]+B[+k0])*(B[]-B[+k1]))`.

Figure 4.16 Variations of the STREAM "add" benchmark with increasing number of elements accessed at an offset.



Addition and multiplication of three arrays and accessed with symmetric offset `A[]=A[]+c*((B[+k]+B[])*(C[+k]+C[])-(B[]+B[-k])*(C[]+C[-k]))`.

Figure 4.17 Variations of the STREAM "add" benchmark with increasing number of fields and elements accessed at an offset.

The error between measurement and model seen in the above examples is very reasonable. The jumps in throughput are captured well and no striking deviations have been observed. The errors most relevant for the actual FDM loop kernel, which is served from main memory and hence corresponds to the longest loop lengths and largest offsets, did show a dependence on certain loop kernel features. However, the error is still far from the one observed for the full FDM loop kernel. Naturally the above examples could be further expanded until eventually the full kernel is obtained, but the number of possible variations is much too large to be fully analyzed. The bottom up approach is therefore not pursued further and other techniques are used instead. The findings from the above analysis are valid nonetheless. For one thing the ECM can very accurately model these simple loop kernels. For another, there were no significant jumps in the error, but instead a gradual increase with increasing kernel complexity. This theme continues on in the following investigation.

4.4.2.2 Top down

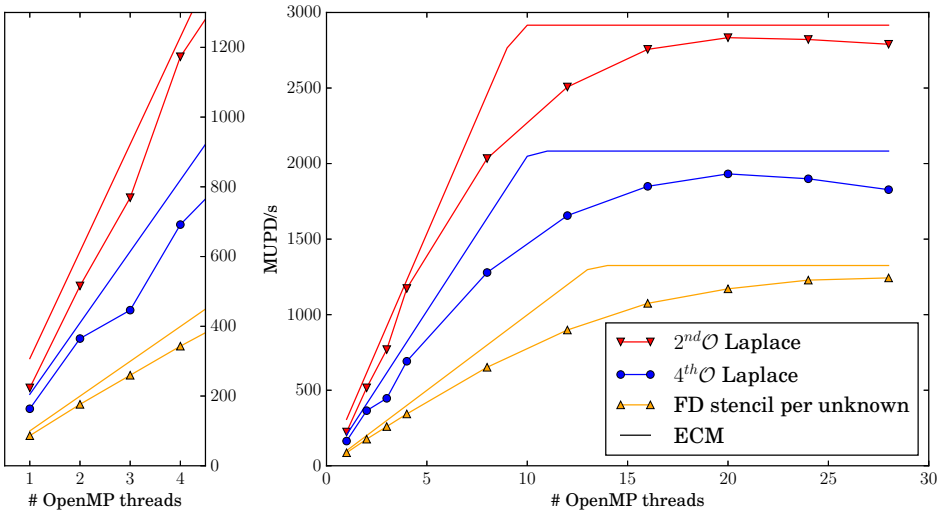
For the top down or drill down analysis the full FDM loop kernel is investigated with the aid of the builtin performance counters. They are used to find discrepancies between the model input parameters and their actual values, as well as verifying its output. By repeatedly refining the measurements it is possible to close in on the problem incrementally. When a bottleneck is uncovered, countermeasures can be employed. The biggest limitation with this analysis is that the performance problems are specific to the code, the compiler and the platform on which it is analyzed. It can also be very challenging to determine any issues in the first place as the performance counter give very basic measurements that can combine a multitude of different effects. Additionally the counters differ slightly between hardware versions and often references have to be established for normal operation. This is the reason why a mixed approach between the bottom up and top down technique is used here. Several intermediate loop kernels between the basic benchmarks from the previous section and the full FDM loop kernel are utilized. Each

of these is in turn analyzed with a basic drill down procedure. This highlights which of the major tuning techniques impacts the performance model accuracy most and what problems they introduce. Finally the full kernel is analyzed in even more detail to further improve its configuration.

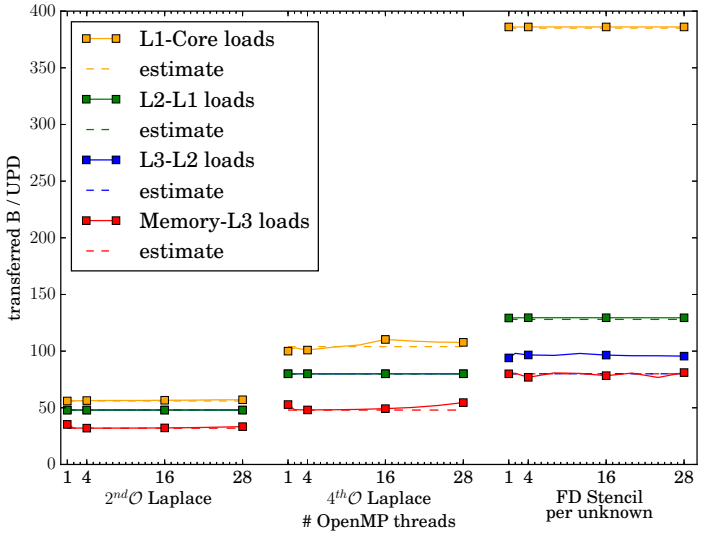
The initial loop kernel is the "add" case from the STREAM benchmark. The next step is a Laplace operator stencil, which already incorporates the star shaped stencil structure that is also found in the FD stencil [27, 78]. However, it operates on a scalar field and the second order version has a span of two, thus avoiding many of the memory access complexities. As the next step the Laplace operator is used in a fourth-order configuration, thus increasing the stencil width to the full size. Finally the actual FD loop kernel is used, increasing the complexity to four fields in a AoSoA memory layout and coordinate splitting. In parallel the influence of the spatial blocking is analyzed.

The comparisons between measurements and models are shown in Figure 4.18a for the different stages. The relative error of the model compared to the measurements using all cores increases with the complexity of the loop kernels. The triad benchmark [130] (not depicted) is within a relative error of 0.4 %, for the second order Laplace operator it is 5.2 %, for the fourth order Laplace operator 14.0 % and finally for the full FD semi-stencil it is 6.6 %. This indicates that there is not a single issue causing the estimate quality degradation, but instead every increase in complexity can lead to new artifacts not considered by the models. This is not surprising, considering the extreme complexity of modern CPUs with many interacting performance enhancing features.

The most influential aspect in the ECM model is the correct number of cache line transfers between the various levels. For the stencil based codes, operating directly on structured grids, it can be determined through the layer and line criterion. For the FD version without spatial blocking the domain is of size $198400000 \times 1 \times 1$, which guarantees that there are no unexpected cache hits. The AoSoA memory slice size is left at 1328 in order to still be able to fit the coordinate split loop kernel in the L2 cache. To verify whether these requirements are met, the transferred data between the cache levels is measured using the built-in hardware counters. LIKWID [147] is used to collect statistics on cache line loads and evictions and they are compared to the theoretical values. This is visualized in Figure 4.18b for the three kernel flavors. Only the loaded amount is shown, since the stored amounts showed good agreement for all versions. The most apparent error lies in the MM transfers with a single core. This is due to a limitation of the performance counters, which can only count memory transfers on sockets on which a monitored thread is running. The amount is therefore duplicated, which is not very accurate as the used memory may be distributed arbitrarily between the sockets, but it is given for completeness. For core counts > 1, the estimates of the second order Laplace operator match the measurements very well. For the fourth order Laplace operator some small deviations arise in the L1 cache and MM transfers for large number of threads. The increase in memory to L3 cache loads can be explained by the shared nature of the L3 cache. The cache has a size of 35 MB per socket, which provides plenty of cache for small thread counts, but only 2.5 MB per thread when all cores are used. The restricted per-core cache, more than expected overhead and an in practice not ideal cache behavior in turn lead to more evictions to and reads from memory. The differences between estimate and measurement become significant for the FD stencil. While the MM-L3, L2-L1 and L1-core transfers are still well predicted, the amount transferred from L3 to L2 cache is larger than expected. The differences correspond to 8 additional values loaded per update. The counts in



(a) ECM models and measurements



(b) Transfer measurements

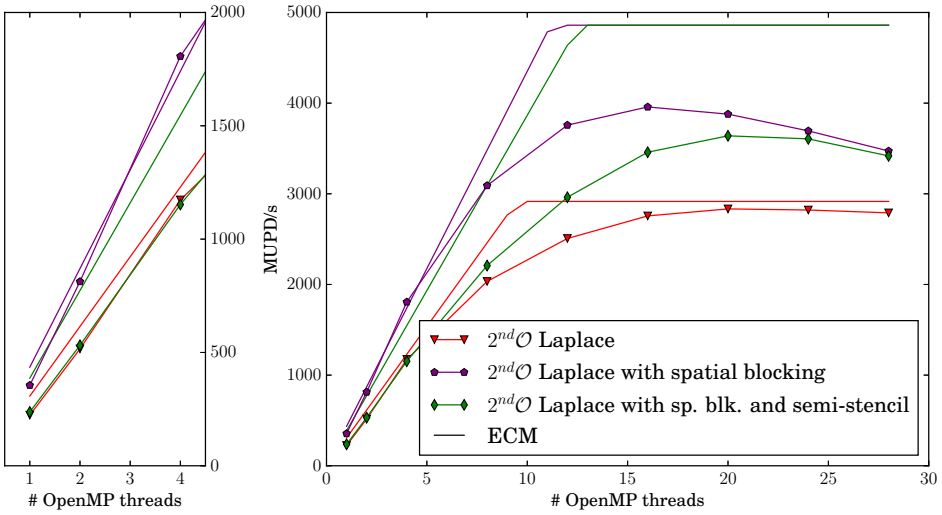
Figure 4.18 ECM estimates compared to measurements of stencils of varying complexity. The FD stencil incorporates CS, but no spatial blocking and is given per unknown for scaling purposes. Adapted from [157].

Table 4.8 are underestimating the number of points transferred in the y- and z-loop by one point each. Tests with different memory slice sizes show that the center point of the stencil evaluation, which should remain in the L2 cache after the loop in x-direction, has been evicted to the L3 cache and needs to be loaded again. The cache lines are being evicted by a cache feature that is not accounted for in idealized model, such as the in practice often used least recently used replacement strategy and unfavorable memory addresses that overburden the 8-way associativity. It is important to note that this effect only arises in combination with the coordinate splitting.

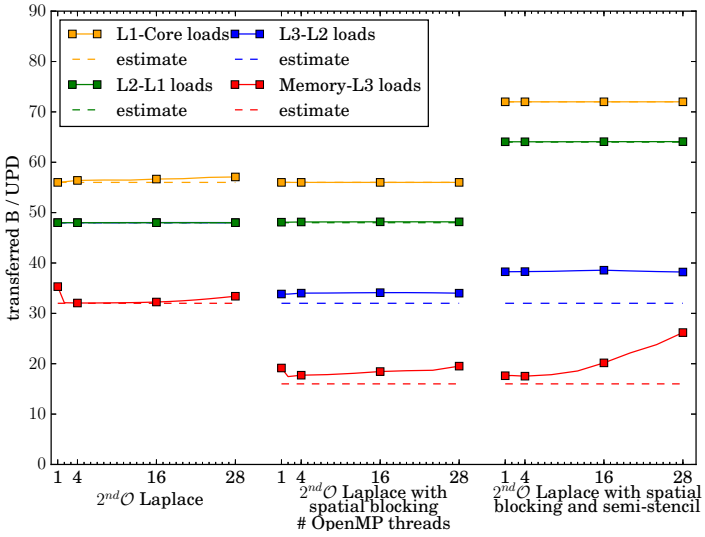
Another optimization feature, which is used by the fully tuned FD loop kernel, is spatial blocking. It significantly alters the cache usage and is therefore crucial in regards to the ECM model. Again the model results are contrasted with measurements, this time for benchmarks of the second order Laplace operator, one in a straight forward AoSoA implementation, one employs spatial blocking and a third one uses spatial blocking and coordinate splitting. The results are shown in Figure 4.19a. The second order Laplace operator is used for this comparison, as it exhibited only very small estimate errors in the previous test. This ensures the measured error really stems from the spatial blocking and has a similar stencil structure to the FD kernel at the same time. The modeling quality deteriorates significantly when moving to the kernel with spatial blocking. Using all cores the relative error is again 5.2 % without blocking. However, with blocking it increases to 40.0 % and with the addition of a semi-stencil it further increases to 42.2 %, which is sufficient to question usability in many applications.

Smaller core counts paint a different picture. The version of the second order Laplace operator with spatial blocking and to a lesser extent also the version with spatial blocking and semi-stencil exhibit the expected increase in throughput for increasing number of threats. However, between 16 and 20 threads, well into the bandwidth limited regime, the throughput decreases again. The reason for this peculiar behavior is a shared resource that introduces increasing overhead when the number of threads exceed its capacity. An investigation into the limiting resource is provided later, along with the performance counter figures. Unlike for the bandwidth limited regime the model and measurements match fairly well in the compute bound regime. The exception to this is the version of the Laplace operator that includes the semi-stencil. This is a strong indication that much of the modeling inaccuracies originate in the complexity of the loop kernel's division into multiple loops.

Further reasons for the offset between measurement and estimate are analyzed through the use of the hardware performance counters. The transferred bytes per update are visualized in Figure 4.19b for the different cache levels. The plain second order Laplace operator shows very good agreement as has been determined before. The spatially blocked version on the other hand exhibits significant deviations. While the L1- and L2-cache transfers are still very close to the predictions, this is not the case for the L3-cache and main memory loads. The L3 cache transfers increase is independent of the core count, while the main memory increases slightly with the number of cores. The loop kernel with spatial blocking and semi-stencil exhibits the same deviations, but more pronounced. Especially the main memory transfers increase significantly and almost linearly for more than ten threads. As before this can be attributed to a cache behavior that differs from the assumption of a perfect caching strategy. The therefore ill-fulfilled layer criterion leads to spills into larger cache levels. This is not too critical for the L3 cache at lower core counts, as each core has additional space available. However, it becomes problematic when the full CPU is utilized and leads to the observed deviations and performance breakdowns. The increase in data loaded from the L3 cache corresponds to one additional value (8 Bytes) compared to the



(a) ECM models and measurements

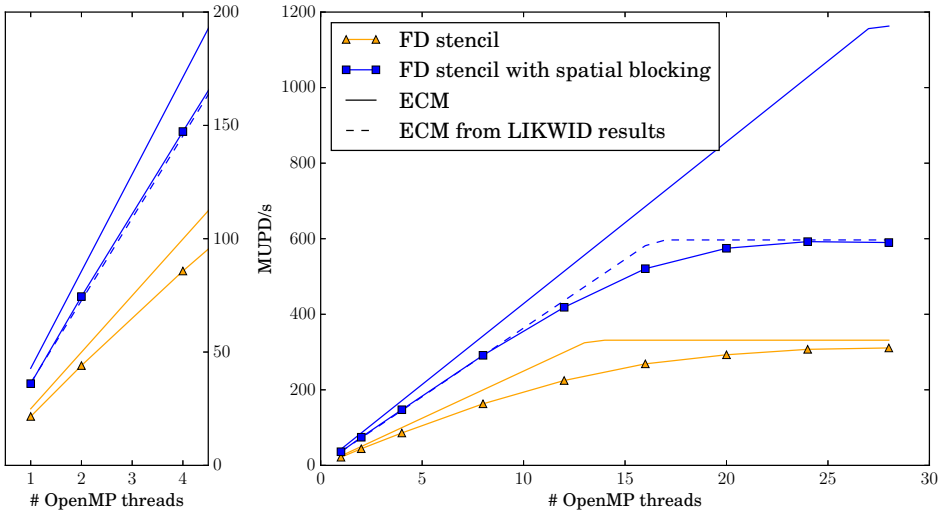


(b) Transfer measurements

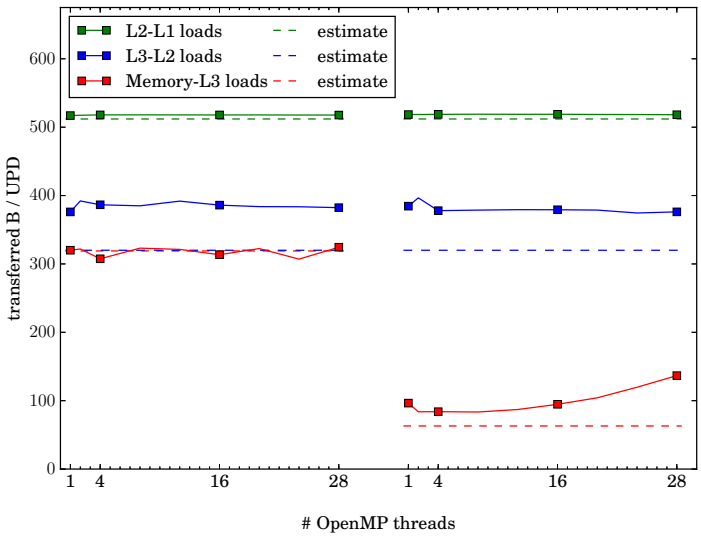
Figure 4.19 ECM estimates compared to measurements of the second order Laplace operator with and without spatial blocking. Adapted from [157].

number determined through the line criterion. This is almost identical to the effects observed for the FD stencil without spatial blocking.

The issues detected in modeling the second order Laplace operator and enhancing it with coordinate splitting and spatial blocking are also present in the full finite difference stencil. The estimates from the ECM model and the measurements for the full FD stencil are given in Figure 4.20a and show an error of 6.6% and 70.4% for the version without and with spatial blocking, respectively. The transferred bytes per cache level, shown in Figure 4.20b, exhibit the



(a) ECM models and measurements



(b) Transfer measurements

Figure 4.20 ECM estimates compared to measurements of the FD stencil with and without spatial blocking. The L1 cache to core transfers are omitted as they show excellent agreement between estimate and measurement. Adapted from [157].

superimposed errors from the wide stencil span and complexity, shown in Figure 4.18b and the spatial blocking of the simplified kernel in Figure 4.19b. Particularly detrimental to the estimate

accuracy is the almost doubled number of reads from main memory. The ECM model predicts hardly any bandwidth limitation for the full FD stencil, but due to the increased memory interface traffic this barrier is easily reached in practice. In order to further analyze this, the ECM model was enhanced with the values from the transfer measurements. The dashed line in Figure 4.20a shows excellent agreement for the bandwidth limited core counts. This shows that the ECM model works accurately if the input is exact, but since the model was calibrated with the same measurements it is reproducing, no further insight is gained from such an analysis.

The full FDM kernels with and without spatial blocking are further analyzed to uncover any additional reasons for time loss. For this the stall cycles are measured. Stall cycles occur when no productive operations can be completed because some resource is depleted. Naturally it is of interest which resource is causing the stall cycles. All executed cycles are separated into four categories following [65]. The walltime spent on each of the categories per update is visualized in Figure 4.21. The four categories are front end (FE) bound, back end (BE) bound, bad speculation

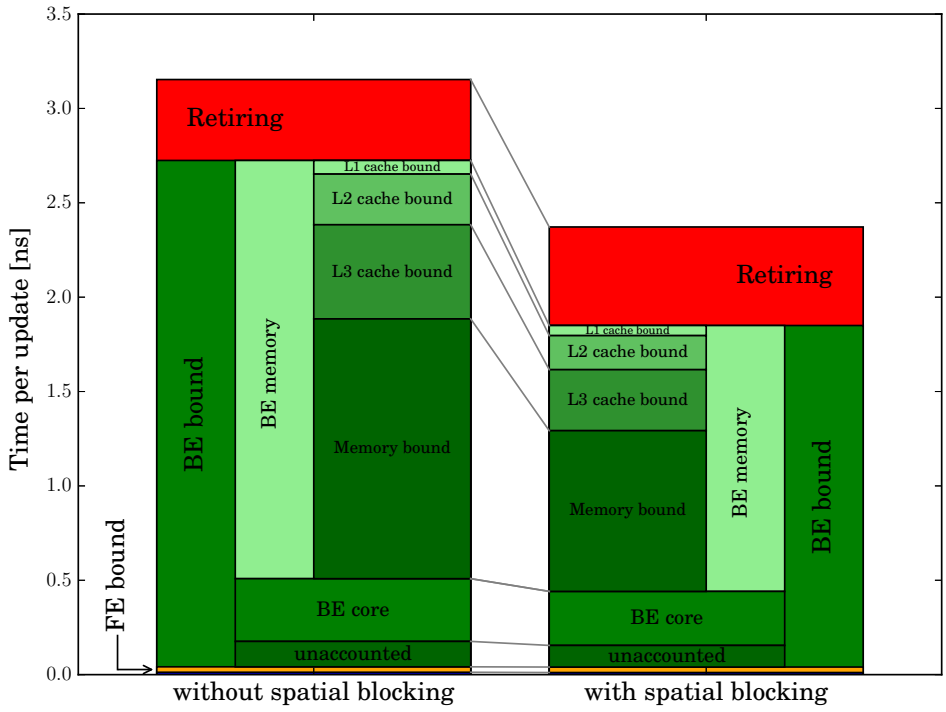


Figure 4.21 Breakdown of the time for a single FD update spent on processing instructions (retiring) and stalls due to resource depletion.

and retiring. Stall cycles due to bad speculation are caused by branch misprediction. For reference see the introduction of branch prediction features of modern CPUs in Section 4.1.1. It allows the CPU to load and execute code speculatively by what it deems the most likely execution path. In the case of a misprediction all instructions executed ahead of time have to be discarded and thus cause stall cycles through the unnecessarily executed instructions and the waiting time until the proper resources are loaded. Fortunately this effect is negligible for both kernels. It constitutes less than 0.5 % of the stall cycles for the spatial blocking version. The second category, depicted in yellow in Figure 4.21, encompasses the duration the kernel is front end bound. For

the spatial blocking version this is 6 % of the stall time and does not have a relevant impact on the performance. Stalls due to the front end occur when the OOO-execution engine is not provided with instructions fast enough, causing a starvation. The reasons that can lead to such a situation are manifold, but as it is not relevant here they are not investigated further.

A significant time of an update is attributed to the retiring category. This category summarizes all instructions that performed productive work. This is measured by counting the cycles in which instructions executed in the OOO-engine are accepted for retirement and thus contribute to the advancement in code. A high retiring rate is therefore important for high performance. However, this metric has to be used with care, as a high retiring rate only guarantees many successful operations, but it does not include a measure of how much these instructions contribute to the actual progress in the simulation. This is an identical difficulty as with the cycles per instructions performance figure. An extreme case would be the NOP (no operation) instruction, which counts into the retired operations, but does not contribute to the advancement of the simulation at all.

The final category is that of stalls due to the back end. It makes up most of the stall cycles, which is to be expected when operating in the memory bound regime. The back end bound stalls can be further subcategorized in core and memory origin. The BE bound stall cycles due to the core are the result of advanced core internal techniques, such as store forwarding, limiting the throughput. These stalls occur at levels very close to the core and do not depend on the memory behavior. All main memory related behavior is included in the memory BE bound measure and this is what the spatial blocking is addressing. By reducing the memory and cache transfers the cycles spent waiting for these resources is cut down significantly. A closer look at the exact distribution of the memory BE bound stall cycles reveals that the ones originating in main memory make up more than half the cycles. They also see the biggest reduction when spatial blocking is introduced. As this part has the smallest bandwidth and highest latencies, the impact on stall times is largest. Due to blocking the stall cycles originating from the L3 cache are also reduced significantly. This matches the premise of reduced L3 cache usage. The L1 and L2 cache induced stall cycles are also reduced very slightly, which, just as the slightly increased retiring cycles, can be attributed to higher code complexity in handling the spatial blocking and which occupies the core while the caches load data.

The in depth drill down analysis has confirmed the findings from the previous analysis. Most stall cycles are induced by memory and cache transfers and all further causes are of little relevance. Paired with the increased transfer volume detected earlier, this is the main cause for the deviations. These difficulties do not only cause performance issues, but also impact the model accuracy.

In conclusion there is no single issue or technique that has been applied and which caused the majority of modeling uncertainty. Instead every bit of complexity that is added to the code introduces another small error. This accumulates with increased tuning and eventually causes the performance estimate to reach unacceptable levels of error or prohibitively large efforts in further refining the model.

4.4.3 Additional tuning refinements

The tuning in the previous sections has lead to significant performance improvements. Through the use of performance models it was furthermore discovered that there are still some inconsistencies between the anticipated and the measured performance. In the following these shall be addressed to make the best use of the hardware. As all optimization and analysis tools are already

in place this can be achieved with little effort. In general however, this complex setup is only practical when very large gains can be expected and the pay off is ensured by long simulation duration.

The first approach exploits the knowledge about the discrepancy between the expected and the actual memory transfer sizes. In order to increase the performance the actual cache and memory transfers have to be reduced to the theoretical values. This is easily influenced through the block sizes. Starting with the block size in x-direction or memory slice size, the transfers up to the L3 cache can be controlled. The block size is successively reduced until the measurements for the L3 cache match the expected values. The largest block size that maintains transfers within an acceptable range is 976 points. The measured transfers per update are then 520.4 B/UPD for loads from L2 cache and 326.2 B/UPD for loads from L3 cache. These are reasonably close to the theoretical values of 512 B/UPD and 320 B/UPD for L2 and L3 cache, respectively. The next step is adjusting the block size in y-direction. Except for very small sizes this has no influence on the cache transfers up to and including the L3 cache. Instead, it has a direct impact on the data transferred to and from main memory. As there is only very limited bandwidth available for these transactions it is particularly important to minimize this aspect. The minimal memory transfers for an x block size of 976 is at a y block size of 7 points. The data read from main memory is 98.1 B/UPD, which is still considerably higher than the ideal 64 B/UPD, but also significantly less than the 136.7 B/UPD that were the result of the block size selection according to the layer criterion. These reductions pay off in an increased performance of 689.7 MUPD/s running on all 28 cores and is depicted as "B" in Figure 4.22 and Table 4.12.

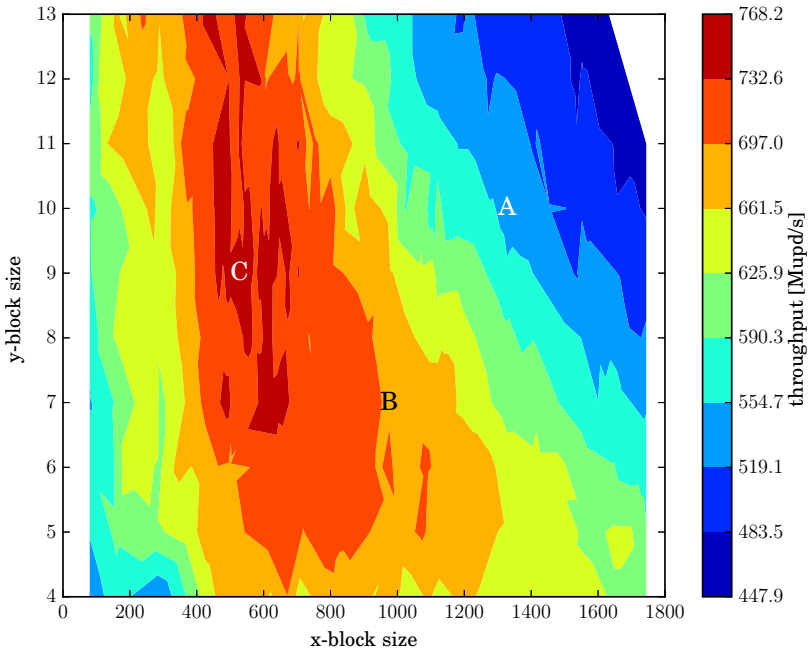


Figure 4.22 Measured throughput for a range of x- and y-direction block sizes, with parameters based on theoretical cache use (A), measured transfers (B) and timing measurements (C). Adapted from [157].

There is the possibility of even better performance when the block sizes are not systematically tuned to the predicted values, but instead can be chosen freely. This opens up the possibility to use block sizes with mixed cache usage per stencil direction. The inner finite difference points in the y-direction could for example be reused from L2 cache while the outer points are loaded from the L3 cache. Taking all possible combinations into account is beyond manual analysis. The space of potential configurations is therefore tested via a brute force approach. Figure 4.22 shows the measured throughput depending on the block size in x- and y-direction. The configuration with the highest throughput is marked as "C" and uses an x-block size of 528 points and a y-block size of 9 points. The measured throughput is then 768.2 MUPD/s.

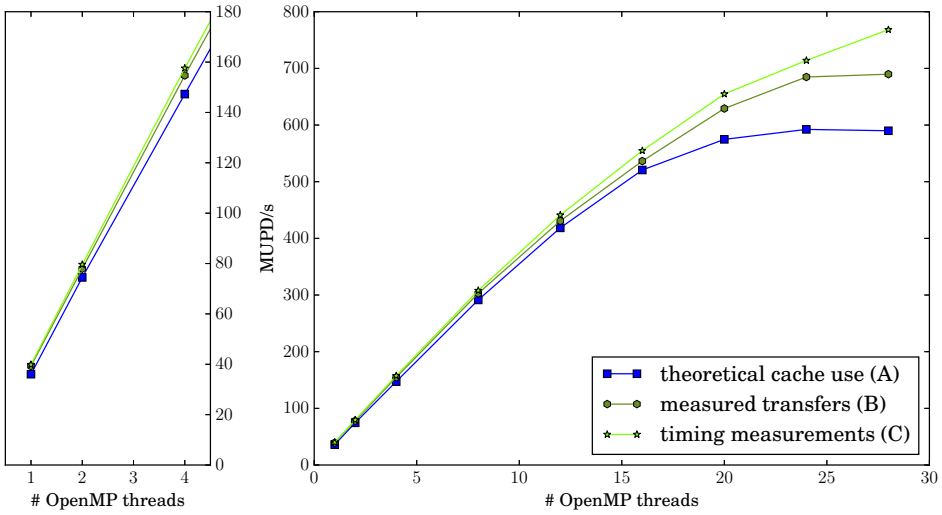


Figure 4.23 Performance scaling of code with parameters based on theoretical cache use (A), measured transfers (B), timing measurements (C). Adapted from [157].

The experimentally determined block size selection shows that the safety margins in choosing the block size from the line criterion have been too small. The optimization by transferred data rather suggests that 64% of the theoretical line size are a suitable choice. The freely optimized block size leads to an even larger margin as it utilizes only 34% of the original line size. All three different configurations are compared in Figure 4.23 from a scaling perspective. For a lower number of threads hardly any difference can be observed, which is expected as the memory and cache bandwidth is not a significant limitation. This changes for large core counts when bandwidth becomes a limiting resource. The improved configurations (B) and (C) clearly make better use of the available bandwidth and thus shift the balance back towards the arithmetic limitation or, in other words, adjust the codes compute to bandwidth balance towards the balance offered by the hardware. This reduces the gap between the behavior predicted by the original ECM model and the measurements. While the performance of (A) was overestimated by 97.1%, it reduced to 68.6% and 51.4% for (B) and (C), respectively. It should be noted that the ECM model was not adjusted to the changed block size and the changed cache usage for configuration (C).

parameter choice by	row-/layer-criterion configuration	optimized transfers	optimized throughput
x-direction block size	1328	976	528
y-direction block size	10	7	9
million updates per second	517.4	689.7	768.2

Table 4.12 The performance of the FD loop kernel for different tuning parameter choices.

4.5 Performance optimization and modeling conclusion

In this chapter the iterative process of performance optimization and model refinement was applied to the FDM and LBM loop kernel. Through this process the performance was steadily increased to over four times the throughput of the basic parallelized versions introduced at the beginning of Section 4.3. At the same time several different performance models were applied with varying levels of precision. Focusing on the models first, it can be concluded that a very wide range of accuracy can be obtained with an equally wide range of modeling options. These options also vary greatly in the required knowledge and effort. It is therefore that all of the presented models have their merit, but for each task the right tool must be chosen and there are limits to the accuracy that can be achieved within reasonable work. The basic roofline model, as introduced in Section 4.2.1, is not very accurate and was found to overestimate the performance by more than a factor of three. This makes the estimates very unreliable. However, a great advantage of this approach are the few prerequisites. This allows predictions purely based on assumptions about the code and the specifications for the intended hardware. A rough estimate can therefore be made without implementation or using hardware. A further advantage is the easy extension of the roofline model. As more detailed information becomes available it can be incorporated in the model. The measurement enhanced roofline model, introduced in Section 4.2.2, uses sustained bandwidth and tool based code analysis to great effect. The improved model was found to be significantly more accurate, even though occasionally the performance was still overestimated by more than a factor of two. A drawback is, that in order to generate the measurements the code must have been at least partially written. At least the loop kernel is required for the software side and meaningful measurements of the sustained bandwidth are necessary on the hardware side. Nonetheless, there is still no intricate knowledge required, neither about the compiled code nor the hardware. The ECM model differs significantly in this respect. While it is essentially a further extension of the roofline model, it adds factors that require detailed knowledge of the memory and cache hierarchy. These go beyond the information published by the manufacturer and need to be supplemented through reference measurements. Also an intricate knowledge of the code and the potential caching behavior is necessary. This makes it intricate to apply the ECM model, but the results can be highly accurate. Especially the scaling in the compute bound regime was generally estimated well for loop kernel with a low complexity. For highly optimized and complex kernels however, the predictive qualities degraded quickly. In the bandwidth limited regime the ECM model uses the same approach as the roofline model with sustained bandwidth and thus is subject to the same levels of accuracy.

Naturally the process of refining the model with more properties and increasing the accuracy by measuring more components can be continued until the desired results are obtained. However,

at some point this defeats the purpose of the performance models as eventually all abstraction has been eliminated or the model only reproduces the measurements with which it was calibrated. Additionally the effort necessary to improve the accuracy grows quickly, while the returns in better results diminish. Despite occasional problems with the accuracy of highly optimized code, the performance models are important for the tuning process. For example the deviations between model and measurements can also be caused by unknown inefficiencies in the code, rather than shortcomings of the model. Investigations into the source of the discrepancy can therefore benefit the code as well as the model. The tuning steps taken in this work were guided by this idea and the performance modeling lead to an improved code design that makes better use of the available resources.

However, for the fairly complex FD stencil with numerous optimizations the relative error between measurement and performance model grew to 70.4% and the model did not capture the effect responsible for the performance limitation. Much effort and in depth knowledge on code, compiler, hardware and performance models was required to locate the origin of the discrepancies. This can be expected to be even worse when the simulation method itself grows more complex, e.g. beyond a stencil loop iterates mostly linear over the domain.

This complexity was found to play an essential role in the reliability of the performance estimates. The test cases on which performance models are often showcased, such as the copy, add or triad kernels from the STREAM benchmarks, were in excellent agreement with the measurements. Also the simpler stencil based tests, such as the Laplace operators with narrow stencils, yielded reliable predictions when the kernel was applied to the domain in a linear fashion. This changed rapidly with the increase of complexity in the loop kernel itself. This was especially drastic without the tool assisted analysis, as would be the case when designing a new code. The reliability of the estimates was shown to reduce even more severely with the application of optimization techniques that exploit advanced hardware features and in particular the caching system. This lead to a gradual decline in predictive quality, as can be seen in the sequence of loop kernels in Figures 4.18, 4.19 and 4.20. It was also shown that even when a detailed analysis of the software side is available, as used in the ECM model, the application of a model to a particular hardware is challenging. This is largely caused by the many internal tuning features of modern hardware, which is intended to transparently enhance the performance. In the context of manually optimized code these can cause undesired artifacts that are difficult to diagnose. In the future the trend to increase the complexity of these features is likely to continue, as the limit in clock frequencies has been reached. A further indicator is the development of the MIC architectures.

It has also been assessed how much of the expected speed is actually achieved with the FDM and LBM codes and several bottlenecks were discovered in the process. Section 4.4.2 can not only be taken as a search for modeling inaccuracies, but also be read as the process to use when the measurements do not reflect the expected performance level. The outcome of this analysis is either a problem with the code or a deficiency in the model. In order to rectify this, the user can make predictions for specific characteristics, which are then verified using time measurements and performance counters. When the specific cause of sub-par performance is determined, either the code can be reworked or, if it is a hardware limitation, the model can be augmented by this feature. Here both steps were taken by improving the ECM model accuracy and optimizing the tuning parameters to bring the code closer to the desired behavior. However, it requires extensive in-depth knowledge of the code, the compiler, the hardware platform, and performance models, making it necessary to carefully assess whether such efforts are justified. Some of these

circumstances can be massively parallel implementations, where even a small improvement leads to significant benefits or when concrete deficiencies are suspected. However, in some cases it can be the better option to employ a simple model to merely rule out the largest performance degradations. Ultimately the invested effort and pay-off have to be carefully balanced for all of these models.

A further part that highlighted the bidirectional relationship between modeling and optimization were the adjustments to the parameters of the fully tuned FD kernel. For the configuration of the kernel with theoretical block sizes the ECM model overestimated the actual performance by 97.1% and did not predict the resource responsible for the performance limitation accurately. By adjusting the implementation configuration to fulfill the modeling assumptions this error was reduced to 68.6%, i.e. inefficiencies uncovered with the help of the model were reduced. The additional brute force search for optimal code parameters further reduced the overestimation of the ECM model to 51.4%.

Overall the code optimization itself has brought significant performance increases to both implementations. In total the FDM was sped up by a factor of 76 compared to the basic serial version and a factor of 3.1 to the basic parallelized version. With similar techniques the LBM's performance increased by the factors of 39 and 2.1 to the serial and parallel versions, respectively. Considering that these improvements are possible without any alterations to the mathematical aspects of the simulation methods they come without any drawback in regard to the performance comparison. It is clear however, that extensive tuning must be carried out as the speed ups that were achieved here can easily change the outcome of the overall comparison.

Porting the fully optimized implementations to different hardware architectures also yielded promising results. This is in part due to design choices in the optimizations that omit obscure techniques for marginal gains, but instead are flexible to use and thus future-proof the implementation. However, it is obvious that the configuration has to be adjusted even for small architectural changes. Factors like cache sizes, which may even vary within a single CPU generation, have to be accounted for in order to achieve optimal performance. When the hardware architecture is changed significantly merely adjusting parameters is not sufficient. The optimization series shown in Figure 4.12 makes it apparent that every single technique has to be reassessed when transferred to other platforms. Nonetheless, many had a positive impact on the performance, even without machine specific adjustments.

5 Performance comparison

5.1 Comparison introduction

In this chapter the actual performance comparison of the previously introduced methods is carried out, which has partially been published in [158] before and excerpts of which are used in the following. There are many different ways to perform a performance comparison and it can be carried out under many different aspects. This also means that many different results can be obtained and therefore the objectives of this work are clarified in the following. While there have been comparisons between the LBM and FDM before, they were mostly concerned with convergence orders or plain update rates [113, 114, 167]. This can be traced back to the comparisons rooting in the mathematical or high performance/software engineering regime. Here, instead a comparison from an applied engineering point of view is performed. This leads to some significant differences to previous comparisons in that the full simulation approach including boundary conditions and error evaluation is compared. The quantity of interest (QOI) is not in the convergence order, but rather in the achieved error per walltime or, the other way around, the total time to solution (TTS). Additionally, the timings are evaluated at relatively coarse errors, as the uncertainties that come with simulations seldom justify very strict convergence boundaries in an applied setting. Nonetheless the simulations require a quantifiable figure and a robust setup that ensures a fair comparison. The considerations that form the basis for such a fair comparison is given in the following.

It is most important to note that both codes have been through rigorous performance tuning. It is difficult to determine exactly whether both have been tuned to the same level as only a complete analysis of every option could guarantee this. As the gains become marginal after a certain level of performance has been reached and the measurements are subject to some uncertainty at any rate, this is not required here. Additionally some options with low prospects of benefits and that are highly invasive have intentionally not been applied out of flexibility considerations, such as portability to future hardware platforms. Another important consideration stems from the very different approaches both methods have been derived from. Though they behave similarly in many regards a comparison only through the use of convergence orders and throughput does not cover all their characteristics. The only feasible option is therefore a comparison through a series of carefully designed test cases. These examples are limited by a set of stringent criteria in regards to fairness, feasibility and engineering relevance.

The feasibility is the most obvious limitation, as both methods have to provide the means to simulate a particular example to actually be comparable. While this looks obvious on first sight, there are some hidden difficulties. After all the physical interpretation of the example must be identical and which may not be the given in corner cases where different boundary condition types meet. One such case is the top corner or edge of a regular non-leaky lid driven cavity, where a no-slip condition meets the driven lid. In traditional NSE based methods the corner node would be considered part of the lid and be prescribed the lid velocity. For the LBM however the

boundary is in between cells and the diagonally oriented density distributions interact with the lid and wall cells. Not to mention the boundary to first cell center distance would always be halved compared to the FDM. Of course other boundary types for the LBM are possible which act as placed on the cells directly, but they have the drawback of more complexity, higher computational cost, typically lower order and even more uncertain behavior in the corners. These boundary condition restrictions give rise to the modified forms of commonly used test cases that are used in this work.

The point of fairness in the comparison is not as straight forward to argue as the limitation to the intersection of either method's capabilities. This is because most examples can be altered to fit the strength and weaknesses of the measured approaches. If taken to the extreme for most simulation techniques an example can be constructed for which it outperforms all others. In order to ensure the highest fairness, the known weaknesses are avoided and every parameter choice is carried out with respect to fairness.

To guarantee an unbiased view every possible parameter and design decision combination would have to be tested and the performance differences evaluated from the resulting field. However, as each method on its own already offers innumerable configuration possibilities, such as the type of time integration, spatial order, stabilization, the boundary condition implementation and so on, this is infeasible to carry out. Instead of an all-encompassing comparison some design choices have to be made to narrow down the number of parameters. These choices are made for parameters specific to the test case choice only, such as the Reynolds and Mach number or the domain aspect ratios.

The target quantities by which the quality of the solution is evaluated on the other hand are treated differently. The observed quantity and its target tolerance have a direct and significant impact on the simulation duration since slight differences in convergence order create a strong dependence on the target tolerances. Hence, a range of the target quantities are considered that allows the methods to be set in relation to each other regarding the error. Of course this range is limited by practical boundaries, which on the one hand ensure that a minimum of solution steps are performed and on the other hand are within the limits of computational effort and can still count as typical engineering ranges. The choice of the target quantity is test case specific and is discussed for each example separately. They can however be categorized as classical error norms on the domain or specific areas and indirect measures, such as vortex count and position or lift and drag factors.

The third category of comparison considerations are the method specific parameters, such as spatial resolution or stabilization parameters. These method specific and suboptimal values can directly impede either approach. Hence, they are tuned systematically for optimal performance. As the choice of the target quantity can influence the set of optimal parameters this is done for every quantity and error separately. This ensures the absolute highest performance in every scenario. Of course such an extensive process is generally not sensible in practice. Therefore measurements using "intuitive" parameters are also given. They are purely based on user experience and are on the safe side to ensure convergence.

5.2 Test case choice

The test cases used below shall reflect the goals presented above. For this, a series of tests is selected for which the selection process is shown in the following. They are chosen from well known benchmarks and adapted to suit the use here. Some benchmarks with analytical solution are the plane Poiseuille flow [111] or Kovasznay flow [86, 112]. Other comparison options include the LDC [23, 38, 52, 134], channel flow [81, 107] and more complex test cases [129, 132, 137]. Another source are other comparisons for performance and convergence order [8, 37, 52, 79, 113, 114, 129, 167]. Unfortunately most are not suitable due to compatibility restrictions between the LBM and FDM or because the result quality is difficult to quantify. Others however can be adapted and form the basis for the test cases presented in the following.

Ideally all test cases provide a scalar figure by which the result quality can be measured and set into relation to the simulation runtime. Unfortunately there is a trade off between such a simple measure and practically relevant test cases since simulations are usually performed to asses a number of properties. The sequence of cases presented here bridges the gap between this direct assessment and practical relevance. The examples start out with the minimalistic configuration using a single well defined and direct scalar quantity as the quality measure. The complexity is then increased with each test case towards practical relevance with the drawback of less direct quality assessments from having multiple relevant quantities. Some properties may not even be quantifiable at all, but are nonetheless important to the evaluation of the simulation quality.

Naturally all cases are restricted to the intersection of either methods capabilities. This influences a number of factors, the most important being the possible geometries. As both fully tuned implementations employ globally uniform rectangular grids without any building block methods or mesh refinements all domains are rectangular. However, there may be obstacles embedded within the domain as long as they coincide with the global meshes. This restriction is imposed to circumvent the need for immersed boundary conditions which would open up an entirely new chapter of comparison possibilities, necessities and uncertainties. These geometry restrictions seem very strict at first glance, but in practice many flows can be made to suit this form and such domains are not uncommon in literature. Other restrictions to the choice of test cases come with the flow velocity, i.e. Reynolds number and the compressibility effects represented by the artificial Mach number. The LBM and FDM have been designed to work well within particular ranges of these dimensionless numbers. The test cases are chosen to match these ranges for which the best results can be expected. In terms of Re the flow is designed to remain laminar or turn slightly unsteady as in the test case with the von Kármán vortex street. Concerning the Ma number, both codes work best at around Ma = 0.1 where the impact of compressibility on the flow behavior is still small, but excessive acoustic modes are avoided and relatively large time steps are possible. The most involved area to find a configuration that is compatible with either method is the set-up of matching boundary conditions. As noted in Chapter 3 there is a large difference in how boundary conditions are applied for the LBM compared to classical approaches. This means that the boundary conditions have to be chosen carefully to induce identical behavior. Even in the simple case of Dirichlet conditions differences can arise, such as on edges where two conditions with different velocities meet. The value applied to the edge can be prescribed directly for the FDM. This is not possible for the LBM as there is no explicit edge due to the boundary condition effectively at the half way point for the bounce back scheme and due to the indirect prescription of velocities through density distribution functions. The exact conditions to prescribe

on this domain edge are unclear. Finding configurations that impose identical boundaries on the flow is a very important part of this chapter and therefore requires detailed considerations for each test case.

Within the scope of the compatibility to both implementations there are further important aspects that have to be taken into account. All decisions are aimed at an unbiased comparison by avoiding configurations that are particularly challenging to only a single method. This way the test cases not only show the apparent strong suites of either method, but instead explore the transition zone where either approach is competitive. This gives an indication of the turning point at which one method becomes faster than the other. The exact parameters to achieve this behavior are discussed for every case separately as they are tightly bound to the exact circumstances. The examples presented in the following sections are based on each other. The series is starts with three steady-state flows with increasing complexity. The first is a duct flow. More complex BCs and nonlinearities are added in the second case, the lid driven cavity. The third case is a flow around a cylinder with a square cross section in the laminar flow regime. The methods presented here are not explicitly designed for steady state problems and which is why further time dependent test cases are analyzed. Here these time dependent cases are distinguished between unsteady and transient, which refers to the source of the time-variant flow. The term unsteady is used for constant boundary conditions in time that lead to unsteady behavior and transient is used for time dependent conditions. Both versions are based on the flow around a cylinder used for the laminar flow test cases. This way insights gained in the steady case can be carried forward to the time-variant versions.

5.3 Steady-state flows

In the following the three steady-state test cases of the duct flow, lid driven cavity and flow around a cylinder with a square cross section are presented. The third case further acts as a precursor to the related time dependent test cases given in Section 5.4.

5.3.1 Duct flow

The first test case in this comparison is a steady-state duct flow. Due to its simplicity only a minimal set of features are required from both methods. Furthermore the analytical solution is known, eliminating uncertainty in the reference solution.

5.3.1.1 Problem description

The duct, as depicted in Figure 5.1, has an aspect ratio of $L_x = 15$ to $L_y = L_z = 1$. It is simulated at a Reynolds number of $Re = 100$, where $Re = \frac{U_{max} \cdot L_y}{\nu}$. For the fully developed flow state the maximum velocity is reached at the center of the duct $U_{max} = U(x, {}^{L_y}/_2, {}^{L_z}/_2)$, which can be inferred from the analytical solution. The artificial compressibility in both methods requires that the artificial speed of sound c_s is prescribed. It is determined by the Mach number $Ma = \frac{U_{max}}{c_s}$, which is chosen to be Ma = 0.1. This setup is in the incompressible regime, while offering fast convergence to a steady-state solution. By prescribing the maximum velocity as $U_{max} = 1$, the parameter of the speed of sound and the kinematic viscosity are obtained. An overview of all

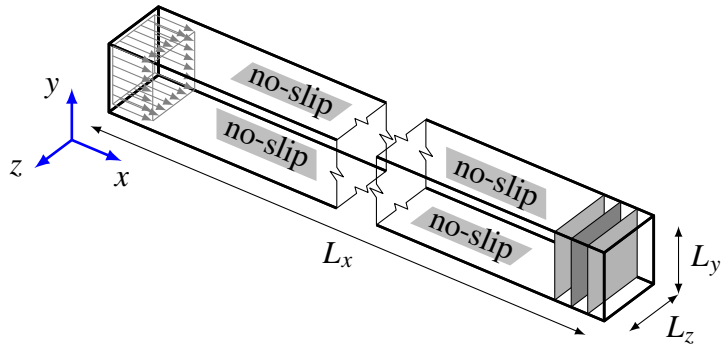


Figure 5.1 The duct flow setup with a block inflow profile and error evaluation cross sections. Adapted from [158].

parameters for the duct flow is given in Table 5.1. It should be noted that for brevity no units are given, but all parameters could consistently represent a real problem and all error evaluations are carried out in this system.

A duct flow requires no-slip boundary conditions along the walls parallel to the flow direction ($y \leq 0$, $y \geq L_y$, $z \leq 0$, $z \geq L_z$). For the FDM the velocity at the wall is simply prescribed to zero and for the LBM the walls are implemented as midway bounce back BCs. The inflow is realized as a block profile. The velocity is smoothly ramped up over the time span t_{ramp} according to the polynomial

$$\boldsymbol{u}_{in}(t) = \left(u_{in} \cdot \left(6\hat{t}^5 - 15\hat{t}^4 + 10\hat{t}^3\right), 0, 0\right)^T \qquad \text{with} \qquad \hat{t} = \min\left(1, \frac{t}{t_{ramp}}\right). \tag{5.1}$$

This polynomial is used as the second derivatives are continuous at the transition to the constant part, unlike the often used sinus based ramping, which caused severe convergence issues for the LBM. The inflow velocity is applied to all inner nodes on the channel side $x \leq 0$. This greatly simplifies the boundary condition for the LB method, since no higher order moments arise, which would lead to transverse velocity components behind the inflow. At the outflow the pressure is prescribed to zero and there is no special treatment of the outgoing velocities.

Parameter	L_x	L_y	L_z	U_{max}	Re	Ma	ρ	ν	$\frac{dp}{dx}$	$U_{in,ideal}$
Value	15	1	1	1	100	0.1	1	0.01	−0.13574	0.47704

Table 5.1 Problem specification of the duct flow.

5.3.1.2 Comparison considerations

The error in the steady state solution is determined through the use of the analytical solution [156]. The infinite series

$$U(y,z) = \frac{4L_y^2}{\mu\pi^3}\left(-\frac{\mathrm{d}p}{\mathrm{d}x}\right)\sum_{i=1,3,5,\ldots}^{\infty}(-1)^{\frac{i-1}{2}}\left[1 - \frac{\cosh\left(\frac{i\pi}{L_y}(z-\frac{L_z}{2})\right)}{\cosh(\frac{i\pi L_z}{2L_y})}\right]\frac{\cos\left(\frac{i\pi}{L_y}(y-\frac{L_y}{2})\right)}{i^3}, \tag{5.2}$$

is truncated and evaluated to double precision. The pressure gradient $^{dp}/_{dx}$ is determined by setting $U(^{L_y}/_2, ^{L_z}/_2) = U_{max} = 1$.

The error is evaluated in the L^2 norm as the integral over the cross section (or the volume of the section). This is a typical choice for engineering applications. As we only have pointwise solutions for the FDM as well as the LBM, this integral is approximated by the pointwise error weighted by the size of the adjacent cells. As opposed to a straight sum over the points this takes into account the half-sized cells near the boundary of the LBM solution. With S as the entire cross section and $S_{adj} = \frac{1}{n_{adj}} \sum S_{cell}$ the cross section fraction of the surrounding cells we obtain for the error

$$\frac{1}{S}\sqrt{\iint_S (\hat{u} - u_{ref})^2 \mathrm{d}y\mathrm{d}z} \approx \frac{1}{S}\sqrt{\sum \frac{S_{adj}}{4}(\hat{u} - u_{ref})^2}. \tag{5.3}$$

A large part of the duct is dedicated to allow the block inflow to develop into the steady-state profile. In Figure 5.2 the error against the analytical solution on the cross section at the x-coordinate is shown. For the comparison the error is evaluated at $x = 13$, where the steady state is reached for resolutions relevant to this test. The correlation between the block inflow velocity

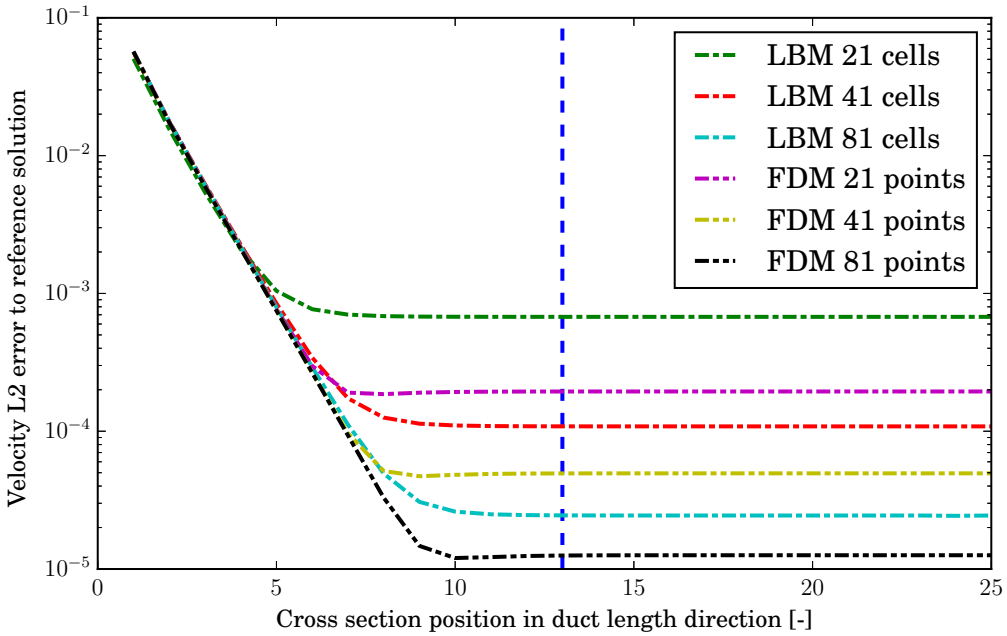


Figure 5.2 Development of error over channel length. Adapted from [158].

and the maximum velocity in the steady flow profile is not straight forward and once again differs for both simulation approaches. The inflow velocity U_{in} is determined in an iterative procedure so that the maximum velocity in the error evaluation plane $U(13, ^{L_y}/_2, ^{L_z}/_2)$ is exactly one.

In addition to the velocity error, the error of the pressure on the cross section and the error in the pressure gradient along the duct are measured. The pressure in the cross section should be constant, hence the pressure error is evaluated as

$$e_p = \frac{p(13, y, z)}{\langle p(13, y, z) \rangle} - 1, \tag{5.4}$$

using a pressure averaged over the cross section $\langle p \rangle$. No absolute pressure value can be used, since the outflow position at which $p = 0$ is not identical for both methods.

The pressure gradient, estimated by the difference in pressure between $x = 12$ and $x = 14$, is independent of the exact position, as long as it is evaluated in an area with fully developed flow. The error is calculated using the analytical gradient

$$e_{\frac{\mathrm{d}p}{\mathrm{d}x}} = \frac{p(14, y, z) - p(12, y, z)}{2} - \frac{\mathrm{d}p}{\mathrm{d}x}. \tag{5.5}$$

Both pressure errors are integrated in the same manner as the velocities.

The total time to solution is determined by measuring the time until the simulations achieve a specified error in the aforementioned L^2 norm. Artificial compressibility methods exhibit a distinct convergence behavior (see Figure 5.3) that is largely influenced by waves traveling through the domain. These waves decay slowly, but cause oscillations in the error. A tolerance is therefore considered reached when the error stays below the threshold permanently.

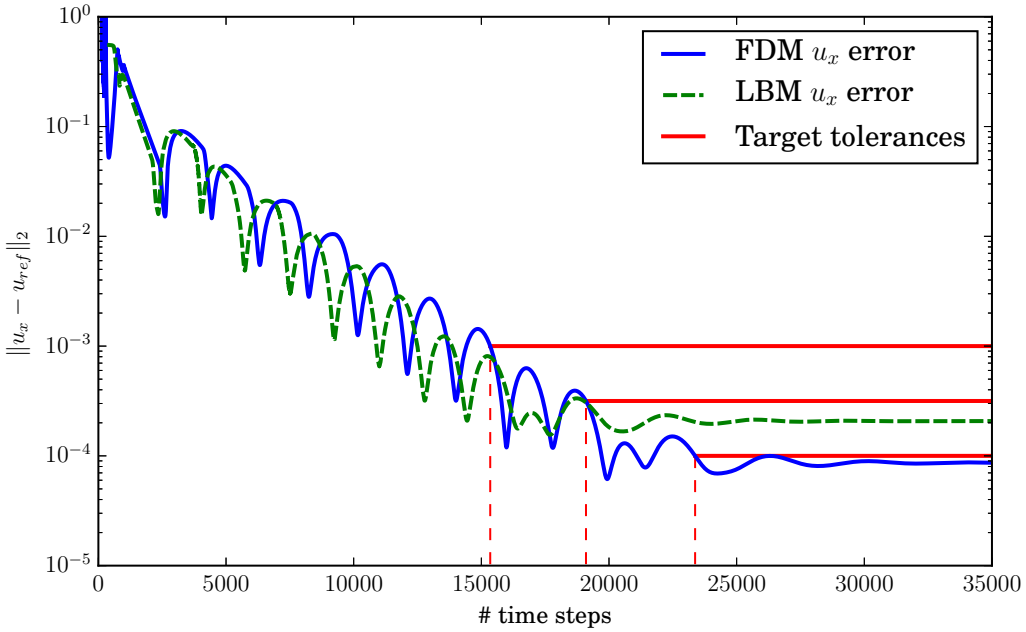


Figure 5.3 Convergence of error in u_x at $x = 13$ over time for FDM and LBM with 31 points/cells over the channel height. Adapted from [158].

The target tolerances are set to 10^{-3}, $5 \cdot 10^{-4}$, $\sqrt{10} \cdot 10^{-4}$ and 10^{-4}, which are typical engineering choices. Any tighter tolerances are usually overshadowed by uncertainty in the choice of material parameters and geometry, outside academic examples [99]. Furthermore their costs are beyond the scope of an optimization process as applied here. A coarser choice of tolerances is excluded, since the simulations do not have to exhibit a proper convergence behavior, but can reach the target by coincidence.

Apart from the geometry, boundary conditions, error evaluation and tolerances, each method has some parameters that do not directly influence the physics. They do however influence the methods' convergence behavior and evaluation cost. Since the best performance is of interest here,

the simulations are tuned for minimal computational cost for every measurement. Simulation runs with different target tolerance can therefore have different optimal parameters.

The optimization is done by an automated process. As time measurements are subject to significant noise, the optimization is instead performed on an estimate of the computational cost, based on the number of time steps and cells/points. Another challenge is the abrupt change from faster convergence to divergence and since it is an integer programming problem. This rules out any common gradient based optimization schemes. A potential option are evolutionary algorithms but the problem at hand features an excessive number of local minima. Configuring these algorithms for reliable detection of the optimal configuration while maintaining a low number of samples requires much experience or experimentation. Instead a combination of simple techniques is used. First the full space of possible configurations is systematically tested with a coarse grid. This is then refined in all potential areas by a bisection strategy. The resulting best configurations are then improved upon with a hill climb technique until no further improvements can be achieved.

Visualizing and interpreting the results is not straight forward, as it is a multi-dimensional space of parameter configurations and simulation durations. The duration is therefore presented for pairwise parameters, as can be seen for the FDM in Figures 5.4a, the LBM 5.4b and also in the Appendix A.1. Of course the duration is not unique if only two parameters are specified. This can be solved by showing the convex hull of the minimum durations. This hides many non-minimal points in between the local minima, but only these are relevant for the tuning. As this strategy would also hide the edge at which the simulation no longer converges, it is detected separately and integrated into the convex hull.

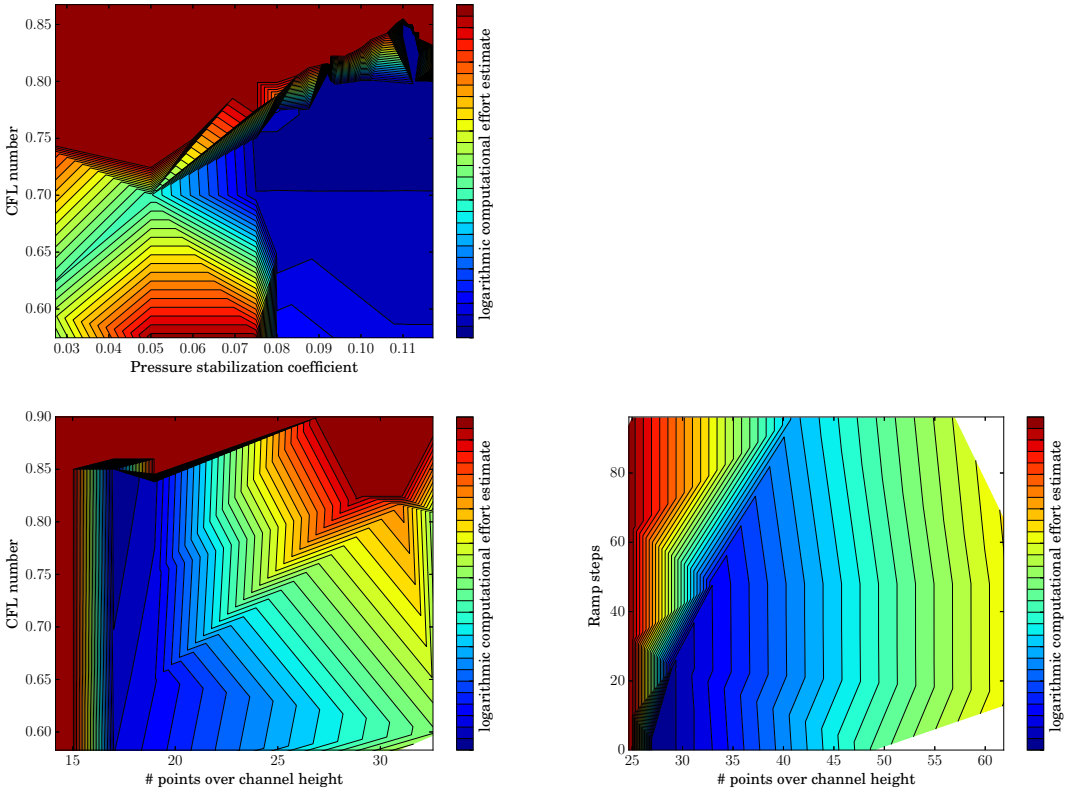
5.3.1.3 Finite difference method

For the finite difference approach the most influential parameters are the number of discretization points and the CFL number. Another parameter is the ramping time t_{ramp}, which determines the duration over which the inflow velocity is smoothly increased from zero. This ramp time determines how sharp the inflow velocity is applied and therefore the amplitude of the artificial wave that moves back and forth through the duct. Furthermore the stabilization parameters for the AV for the velocities c_{vel} and the pressure c_{pres} are optimized. Some exemplary results of the optimization are given in Figure 5.4a and a larger set of combinations can be found in Appendix A.1.

The used effort estimate is the product of the total number of points and the number of time steps to reach the $\sqrt{10} \cdot 10^{-4}$ tolerance. Presented are the two most striking pairs for which the tendency towards lower efforts can be seen clearly. Near the optimum the tested parameters are considerably denser, while at the same time the performance improvements flatten off and sharply changes to divergence. The optimal parameters for all target errors are summarized in Table 5.2.

5.3.1.4 Lattice-Boltzmann method

The LB method requires fewer parameters to be optimized, since no stabilization is required for relaxation times within the stability range. In Figure 5.4b the influence of the two tunable parameters are visualized. The number of fluid cells, specified as the number of cells over the duct height, is again the most influential parameter. The second optimization parameter is the



(a) FDM **(b)** LBM

Figure 5.4 Simulation parameter optimization for the duct flow at $\sqrt{10} \cdot 10^{-4}$ tolerance. Adapted from [158].

Tolerance	10^{-3}	$5 \cdot 10^{-4}$	$\sqrt{10} \cdot 10^{-4}$	10^{-4}
Number of points over channel height	11	15	19	29
CFL number	0.866	0.806	0.838	0.806
Ramp time	3.125e-3	0.0	0.1	0.0
Stabilization pressure	0.11	0.1	0.11	9.195e-2
Stabilization velocity	0.165	0.14	0.14	8.018e-3
Number of time steps	3733	6813	8457	16428
Pressure RMS error	1.594e-7	1.465e-8	6.741e-9	1.675e-7
Pressure gradient L^2 error	2.284e-4	3.518e-5	2.769e-5	5.544e-6
Block size x	24	140	216	24
Block size y	3	19	23	19

Table 5.2 Optimal performing parameters for Fastflo at various tolerances.

ramping time. Since the use of a cosine based ramping causes severe convergence problems if t_{ramp} is not a multiple of the time step size, the polynomial in equation (5.1) was introduced.

Unlike the FDM the equivalent to a CFL number for the LBM cannot be chosen explicitly. The LBM velocities, cell spacing and time step sizes are bound to the used lattice. The factors for conversion between LB and physical dimensions are determined by the geometry for the cell size and the artificial Ma number for the velocity. The time step size is then readily available.

5.3.1.5 Results

The parameters resulting from the optimization process are given in Table 5.2 for Fastflo and in Table 5.3 for ILBDC. The FD method has five parameters that are optimized. Except for the number of discretization points and the velocity stabilization parameter, there is no smooth relation to the target error. This is due to the oscillating convergence behavior demonstrated in Figure 5.3, where a minute change can necessitate another cycle. The measurement of the number of time steps as integers leaves room for slight variations without resulting in an additional step. For this test case the velocity stabilization is, due to the nature of the flow, not required. The parameter has very little influence on the convergence speed, but leads to a slowdown if it is chosen too large. Regarding the LB method, a ramping of the inflow condition is not beneficial,

Tolerance	10^{-3}	$5{\cdot}10^{-4}$	$\sqrt{10}{\cdot}10^{-4}$	10^{-4}
Number of cells over channel height	21	27	33	49
Ramp steps	1	1	1	1
Number of time steps	10365	13780	17166	31667
Pressure RMS error	2.954e-5	1.345e-4	6.996e-6	2.039e-6
Pressure gradient L^2 error	7.950e-6	6.508e-3	1.612e-5	2.440e-6
blocksize	287	233	233	212
other blocksize	1	1	1	1

Table 5.3 Optimal performing parameters for ILBDC at various tolerances.

instead it only delays convergence. It is apparent that the LB method requires more cells and more time steps than the FD method in order to meet the same tolerance. Paired with a mostly lower update rate (see introduction of both codes) the LB method is at a slight disadvantage for large domains or a strict tolerances. For coarse tolerances and the resulting smaller domains however the LBM can perform better as the bandwidth saturation is not as severe and the spatial blocking strategies of the FDM require large domains to take full advantage of the caches.

The measured elapsed times are provided in Table 5.4 and visualized in Figure 5.5. The FD method is significantly faster for the stricter tolerance of 10^{-4}, by a factor of 4.3, while the LB method is 2.1 times faster for coarser tolerances. The source of this difference in scaling behavior lies in the code performance and not the simulation method. The ratio of the number of cells over the channel height that are required for the LBM compared to the FDM reduces from 1.9 to 1.7 towards tighter tolerances and likewise the ratio of the number of time steps decreases from 2.8 to 1.9. The code performance in terms of updates per second exhibits just the opposite behavior and dominates the overall timings. The reasons for the difference in code scaling are, as

mentioned earlier, the spatial blocking in the FDM code which requires large block sizes to reach its full potential and the large memory bandwidth requirement of the LBM which can mostly be served from caches for the smaller domain, but otherwise inhibits the full use of the available peak arithmetic throughput.

Tolerance	10^{-3}	$5 \cdot 10^{-4}$	$\sqrt{10} \cdot 10^{-4}$	10^{-4}
Fastflo	4.431 s	12.172	17.081 s	67.546 s
ILBDC	2.132 s	22.742	56.017 s	288.079 s
Ratio	2.08	0.54	0.30	0.23

Table 5.4 Total time to solution for the duct flow test case.

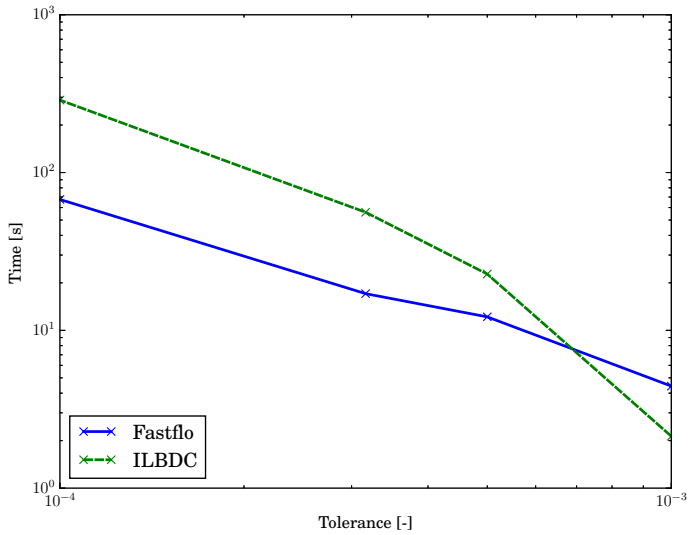


Figure 5.5 Total time to solution for the velocity profile at specific tolerances for the duct flow. Adapted from [158].

5.3.2 Lid driven cavity

The second test case is a flow in a lid driven cavity. It constitutes a significantly more complex flow pattern and has seen a lot of study over the years [134]. As opposed to the duct flow, the convection term plays a crucial role in the steady state solution.

5.3.2.1 Problem description

The leaky lid driven cavity is a configuration that is commonly studied in literature. However, the in- and outflow gap are not straight forward to setup in the LBM context. The complex interactions between the many density distributions on the interface of adjacent BC types require separate handling with complex terms in order to maintain good convergence rates. This contradicts the performance aspect. Additionally it is difficult to keep the boundary condition at the midway point

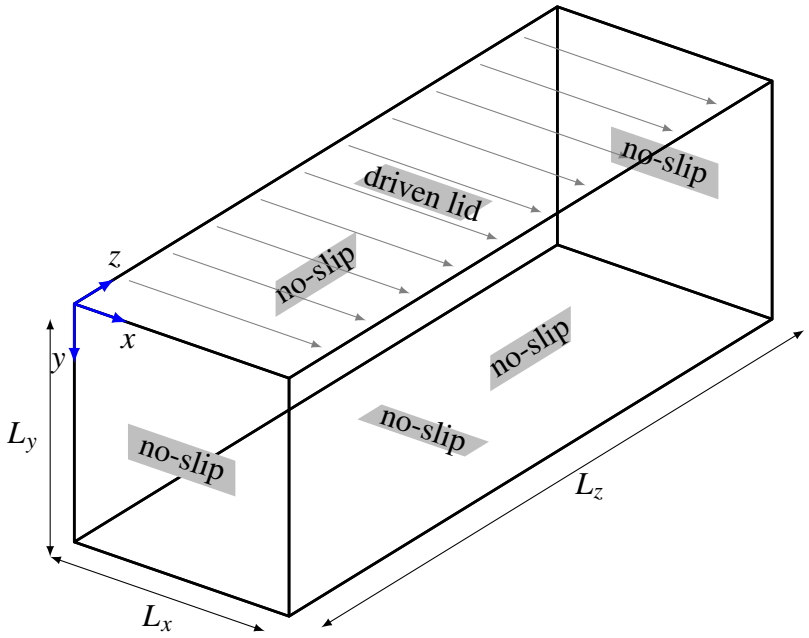


Figure 5.6 The 3D non-leaky lid driven cavity. Adapted from [158].

between cells in this corner case, causing problems in maintaining the gap size and making it difficult to ensure equal behavior in these regions between the LBM and the FDM. An alternative would be using the section between the edge points and their next neighbors as the gap, but this leads to singularities in the corners, just as it does for the non-leaky lid driven cavity. This is of course not suitable for a comparison which also strongly relies on good convergence. However, the non-leaky variant can be modified in order to suppress the singular behavior by prescribing a velocity profile on the lid that naturally tends towards a zero velocity at the edges.

The problem to be solved is illustrated in Figure 5.6. It shows the 3-dimensional non-leaky steady-state lid driven cavity with an aspect ratio of 1:1:3. The lid ($y = 0$) is driven by a Dirichlet boundary condition prescribing the following profile

$$\boldsymbol{u}_{lid}(x, y, z, t) = \left[\frac{U_{lid}(t)}{4}\left(1 - \cos\left(\pi\frac{x}{L_x}\right)\right) \cdot \left(1 - \cos\left(\pi\frac{z}{L_z}\right)\right), 0, 0\right]^T . \tag{5.6}$$

In the LB context non-constant BCs, such as the one above, require advanced schemes. Straight forward application of a bounce back condition can lead to the introduction of spurious velocities through higher order moments, while more complex approaches increase the computational effort. A balance between those two goals is the use of a correction term, such as proposed by [54] and which is used here.

The cavity is simulated at a Reynolds number of $Re = 100$, where the Reynolds number is defined via the maximum lid velocity and the cavity length in flow direction $Re = \frac{U_{lid}L_x}{\nu}$. As before the artificial compressibility nature of the methods gives rise to an artificial Mach number, which is set to $Ma = 0.1$. All simulations are started from a zero initial field and the lid velocities are ramped up to the final value using the polynomial from equation (5.1). A summary of the parameters for the LDC is given in Table 5.5.

Parameter	Re	L_x	L_y	L_z	U_{lid}	Ma	ρ	ν
Value	100	1	1	3	1	0.1	1	0.01

Table 5.5 Lid driven cavity parameters.

5.3.2.2 Comparison considerations

For the error evaluation, no analytical solution is available. Instead a reference solution is computed by performing simulations at a much finer spatial resolution. A comparison of the convergence of both the LB and FD method, as well as an additional finite element approach, is provided in Figure 5.7. The finite element simulation is performed using an implicit steady-state simulation with hexahedral inf-sup stable finite elements of velocity degree 4 and pressure degree 3 and is designed to solve the incompressible Navier-Stokes equation in the most direct fashion. The FEM code is openly available [89].

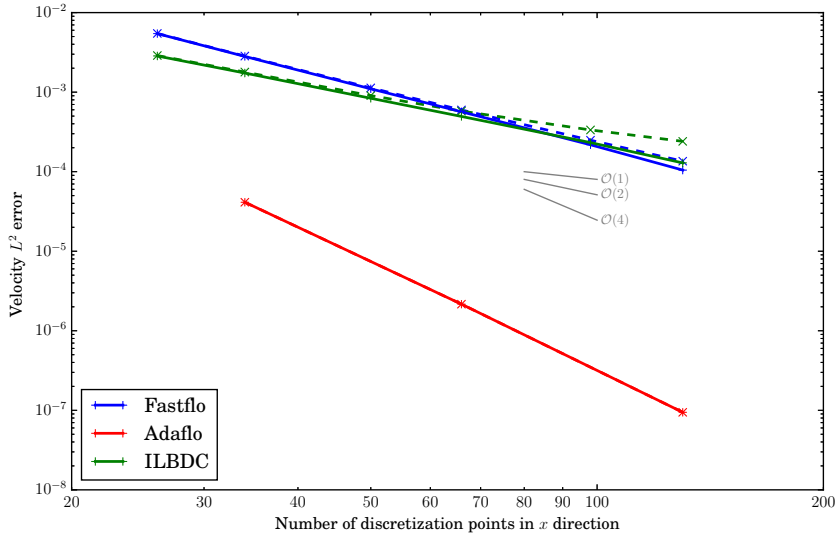


Figure 5.7 Convergence rates of LBM, FDM and the FEM code Adaflo [89]. The dashed lines use the FEM solution as the reference. Adapted from [158].

For each method two separate convergence results are shown. The solid lines are the convergence behaviors that the methods exhibit when each method is compared against its own results on the finest resolution. The dashed lines correspond to the results with the finite element solution as the reference instead.

In the case that each method uses its own reference result, the convergence matches the expected behavior. The finite element simulation converges with fourth order, while the finite difference approach converges at second order. For the lattice Boltzmann method there is no definite expected convergence rate for this case (see LBM description), but it can be up to second order. Hence the measured 1.89 order is a good match.

When the error of all methods is evaluated relative to the finite element reference results, the convergence orders decrease slightly. This behavior is more pronounced at finer discretizations,

which is partially due to the well-known effect of increased convergence rates when comparing to a result with the same dominating error at a marginally higher resolution. Furthermore, it indicates that the methods converge towards slightly different solutions.

For the above case the compressibility exhibited by the LBM and FDM can be ruled out as a cause, since both methods only exhibit compressibility while approaching the steady state. Once this state is established, as in this convergence study, the compressibility effect vanishes completely.

For the actual measurements however, the compressibility when the coarse tolerances of 10^{-3}, $5 \cdot 10^{-4}$, $\sqrt{10} \cdot 10^{-4}$ and 10^{-4} are reached, is still significant. Since this is a characteristic of the methods it is included in the comparison and it is one of the strongest influences on the performance results. Thus it overshadows the differences in reference results. However, since a definite reference cannot be established and to ensure that the methods are treated in fair manner each method will be using its own reference.

As for the duct flow the error calculation has to be defined exactly as it ultimately determines the simulation duration. The ideal approach would be a direct integration of the difference between the solution and a reference over the entire domain Ω

$$\|\boldsymbol{u} - \boldsymbol{u}_{ref}\|_2 = \frac{1}{V_\Omega^{\frac{3}{2}}} \sqrt{\int_\Omega (\boldsymbol{u} - \boldsymbol{u}_{ref})^2 \mathrm{d}V}. \tag{5.7}$$

Since the considered methods do not define a solution between points or cells, a point wise evaluation is preferable. For the FD method this can easily be achieved if the refinements are carried out as powers of two. For the LBM this is unfortunately not possible. Due to the boundary at the half way point in between cells, the refinement levels at which the cells coincide are too far apart for a sensible convergence study. Therefore at least one of the fields in the error calculation has to be interpolated.

In order not to introduce an interpolation error that can affect the accuracy, piece wise fourth degree Lagrangian polynomials are used for interpolation of the reference solution $\boldsymbol{u}_{ref,intp}$. The accuracy of interpolation is three orders higher than the methods' spatial convergence order. Thus the error can be evaluated through a sum over all points/cells in the domain n_p with the difference weighted with the number $n_e^{(i)}$ and volume V^j of adjacent elements

$$\|\boldsymbol{u} - \boldsymbol{u}_{ref}\|_2 \approx \frac{1}{V_\Omega^{\frac{3}{2}}} \sqrt{\sum_i^{n_p} \left[(\boldsymbol{u}^i - \boldsymbol{u}^i_{ref,intp})^2 \cdot \sum_j^{n_e^{(i)}} \frac{V^j}{n_e^{(i)}} \right]}. \tag{5.8}$$

Tests showed that the influence on the error near target tolerances is less than 0.001% for the LDC and thus negligible. Note that the error evaluation for the LBM is carried out in the physical domain, i.e., all results in LB units are scaled using the discrete variables δ_x and δ_t.

Apart from the error in the primary variables the lid driven cavity also offers a second parameter for error evaluation by means of the existence and position of the vortex centers in the mid plane at $z = \frac{L_z}{2}$. Especially the secondary vortices that appear in the corners and near the "inflow"-edge are very sensitive to the simulation accuracy and provide an engineering type measure. As the vortex center position does not typically coincide with the discretization points an iterative scheme is employed to minimize the error. An initial guess of the vortex center is performed by

finding the local pressure minimum. The surrounding 4 by 4 points are then used for fitting a rigid body motion in a least-squares sense to the velocities in the mid plane, which establishes a new vortex center. This process is repeated until the center point is converged. Even with this procedure in place the vortex center position is very sensitive and no uniform convergence is obtained which would allow the result quality to be quantified reliably. Potentially much higher resolutions could stabilize the vortex center convergence, but this would put the simulation runtime outside practical ranges and even more so for the parameter optimization process which requires thousands of instantiations to be simulated. The results are therefore presented for reference and without quantification of the performance.

For the error over the entire lid driven cavity the simulation parameters are optimized for minimal computational cost analog to the duct flow. The target tolerances for the velocity error are 10^{-3}, $5 \cdot 10^{-4}$, $\sqrt{10} \cdot 10^{-4}$ and 10^{-4}. For each tolerance and each method the full set of parameters is optimized.

5.3.2.3 Finite difference method

The finite difference method is again subjected to a large scale semi-automatic parameter optimization with over 100 000 tested parameter configurations. A selection of the results is presented in Figure 5.8 and a larger set is given in appendix A.2. The diagrams only show two of the five

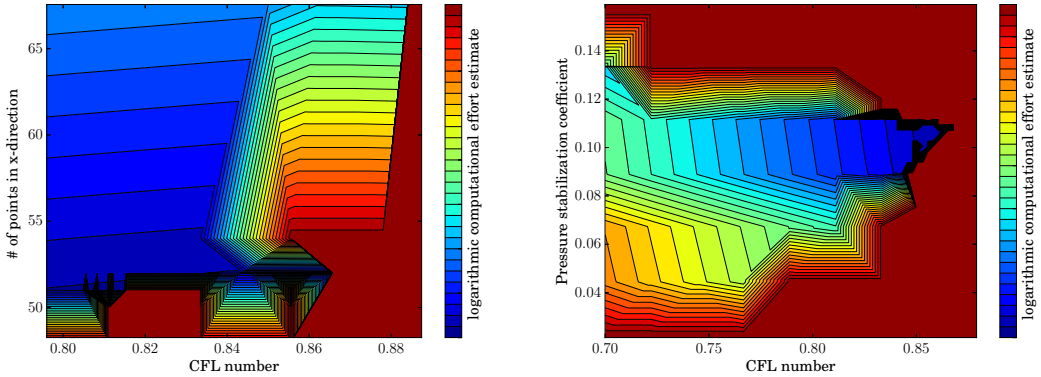


Figure 5.8 A selection of the simulation parameter optimization for the non-leaky LDC at 10^{-3} tolerance with the FDM. Adapted from [158].

optimization parameters at a time, which neglects some of the dependencies. In order to make the trends visible only the convex hull of parameters and runtime is shown. As expected the diagrams show some clear tendencies for shorter runtimes at lower resolution and higher CFL numbers with an abrupt edge when the stability boundaries are reached. The parameters not shown here do not exhibit such a simple relationship and are much more difficult to optimize due to the many local minima. As the performance gains become very small the noise introduced when timing simulation runs is larger and a very good solution is sufficient. The full set of simulation parameter that are the result of the tuning process are given in Table 5.6. The configurations are very similar independent of the target error.

Table 5.6 also contains the results of the code parameter tuning. This tuning is independent of any parameters determined earlier, apart from the domain size and hardware platform. The

Tolerance	10^{-3}	$5 \cdot 10^{-4}$	$\sqrt{10} \cdot 10^{-4}$	10^{-4}
Number of points over lid	53	70	85	134
CFL number	0.866	0.885	0.907	0.913
ramp time	0.0	0.0	0.0	0.0
stabilization pressure	0.109	0.106	0.103	0.1
stabilization velocity	0.0	0.1	0.01	0.0
number of time steps	6025	9078	11696	22072
elapsed-time [s]	15.661	72.056	155.757	996.152
x blocksize	60	76	92	140
y blocksize	8	12	13	20

Table 5.6 Optimal performing parameters for Fastflo at multiple tolerances.

optimal configurations are obtained by brute force search for the code configuration carried out for every target tolerance. This ensures highest performance for all measurements, which are discussed in the results section.

5.3.2.4 Lattice-Boltzmann method

For the LBM only the spatial resolution and ramp time can be tuned, which leads to a significantly simpler search. The results are shown in Figure 5.9 which matches the expectations of lower resolutions being faster, as long as the target error can still be achieved. The number of ramping

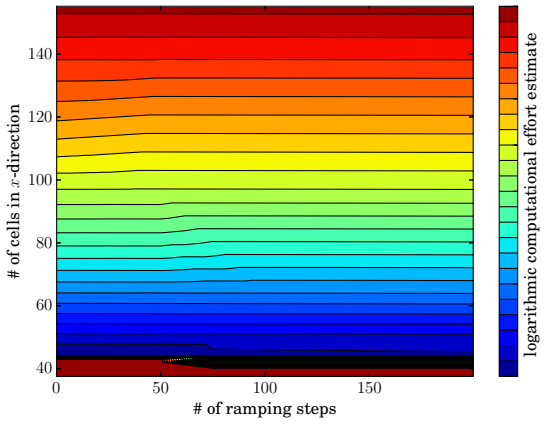


Figure 5.9 A selection of the simulation parameter optimization for the non-leaky LDC at 10^{-3} tolerance with the LBM. Adapted from [158].

steps on the other hand is almost negligible. The lowest overall number of steps is obtained without ramping and for small number of ramping steps the overall number of steps is simply increased by the same number.

The simulation parameter optimization results are summarized in Table 5.7. As for the Fastflo implementation it also contains the code parameter tuning results. These parameters were also

Tolerance	10^{-3}	$5 \cdot 10^{-4}$	$\sqrt{10} \cdot 10^{-4}$	10^{-4}
Number of cells over lid	46	67	85	148
ramp steps	0	0	0	0
number of time steps	9666	15800	21539	43710
elapsed-time (Intel) [s]	20.522	87.122	216.060	2166.567
elapsed-time (GCC) [s]	19.576	89.757	227.266	2186.211
semi-stencil block size	148	256	320	260
addressing block size	1	1	1	1

Table 5.7 Optimal performing parameters for ILBDC at various tolerances.

optimized by a full search of the parameter space. The use of spatial blocking through the indirect addressing did not offer any performance advantages for the settings used here.

The timing results are given for two different compilers, which reflects the findings in tuning the ILBDC implementation in Chapter 4. Only the Intel compiler supports automated use of streaming stores. For a fair comparison the timings with the GCC compiler, which is used for Fastflo and which offers different optimizations, are also given. The result is a small performance difference between both versions, where GCC generates faster code for the smaller domain and the Intel compiler approximately 1% faster code for the large domain.

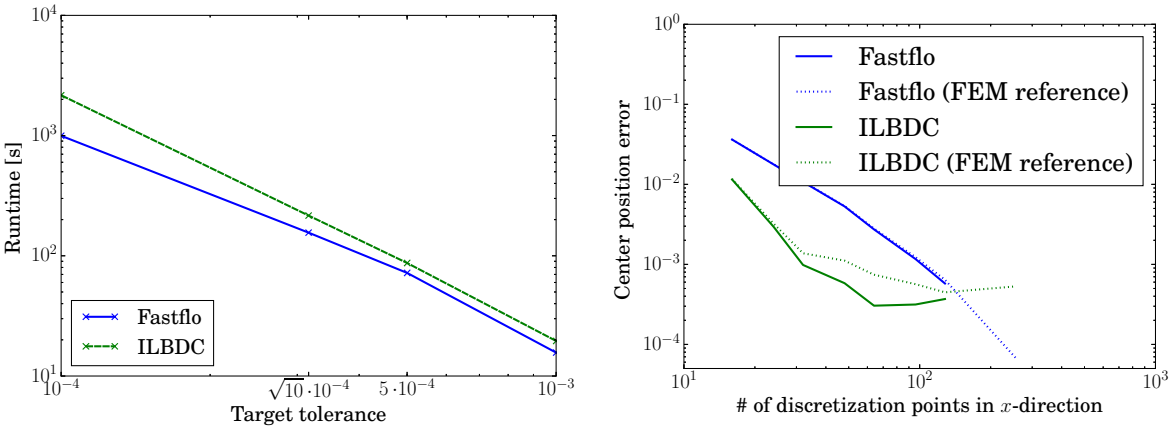
5.3.2.5 Results

The results of the optimization process are given in Table 5.6 and Table 5.7 for Fastflo and ILBDC respectively. Similar to the duct flow there is no direct and smooth relationship between the tolerance and the set of optimal parameters. The required spatial resolution towards stricter tolerances grows more quickly for the LB than the FD method. The same holds for the number of time steps. This causes the performance gap in the actual time to solution, shown in Table 5.8 and Figure 5.10a, to open increasingly. This is similar to the effects that have been observed for the duct flow.

Tolerance	10^{-3}	$5 \cdot 10^{-4}$	$\sqrt{10} \cdot 10^{-4}$	10^{-4}
Fastflo	15.661s	72.056s	155.76s	996.15s
ILBDC	19.576s	87.122s	216.06s	2166.57s
Ratio	0.80	0.83	0.72	0.46

Table 5.8 The total time to solution per tolerance.

The results for the vortex centers are not as conclusive. The center positions in the midway plane of the primary vortex are shown for different spatial resolutions in Figures 5.11a and 5.11b for FDM and LBM, respectively. The FD result shows a clear convergence behavior, which is also illustrated in Figure 5.10b. The LB vortex center positions on the other hand are overall closer together, but eventually fluctuate around an error of $3 \cdot 10^{-4}$. The error is comparable for both methods at the finest used level, but the results are not suitable for a runtime comparison.



(a) Total time to solution.

(b) Convergence of vortex center positions.

Figure 5.10 Comparison of the results for the non-leaky lid driven cavity. Adapted from [158].

Without a clear convergence behavior it could always be outliers, which are being compared and no interpolation of the error to quantify the performance difference can be done. Hence, these indirect solution quality measures are left for the following examples.

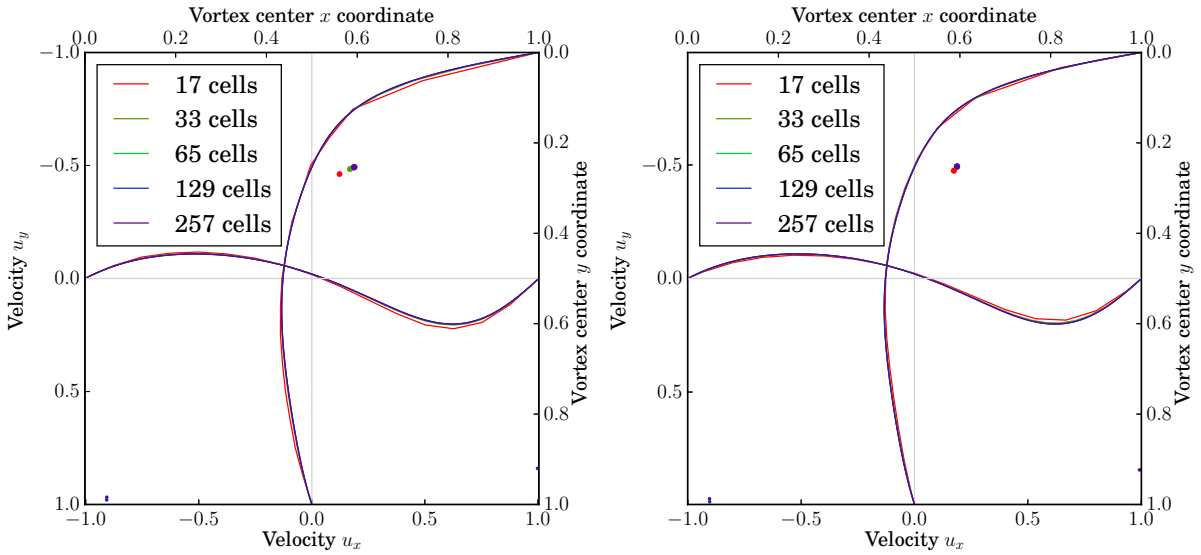
5.3.3 Laminar flow around a cylinder with a square cross section

5.3.3.1 Problem description

In the following the performance evaluations are carried out for the case of an enclosed three dimensional laminar flow around a square cylinder, depicted in Figure 5.12. It is a further increase in complexity through the boundary condition for the obstacle, the resulting flow patterns and the evaluation of lift and drag coefficients. Many variations of this test case have been studied in literature with a variety of configurations, such as open and closed channels, variations in the number and position of cylinders, the cylinder shape and of course a wide range of Reynolds numbers [13, 83, 128, 132, 137]. This test has also been studied in many experimental investigations [84, 115]. The version used here is based on the benchmark in [129] as it best fulfills the requirements for the comparison. The Reynolds numbers are in the range suitable to the two presented methods, the asymmetric obstacle position allows for the lift to be evaluated in addition to the drag and it also offers the extension to transient test cases. Furthermore the given performance figures allow for a general assessment of the expected performance. However, a direct performance comparison is not sensible since the published results were obtained on fairly old hardware and newer publications of the same setup are not available.

The use of the square cylinder allows the obstacle to be realized with a body fitted mesh even with the globally uniform rectangular grid and thus eliminates the need of immersed boundary conditions. These would require a separate performance investigation, due to the wide field of different immersed boundary conditions. Furthermore this test case offers error measures that are closer to engineering applications than errors from analytic flow profiles. All further assessments use the lift and drag coefficients and the pressure difference across the cylinder.

A further benefit of this example is the possibility of using the steady-state version as a precursor to the transient test cases introduced later. Many of the method-specific parameters



(a) Fastflo **(b)** ILBDC

Figure 5.11 Convergence of the vortex center positions and velocity profiles for the non-leaky LDC at Re 100. Shown is the cross section at the mid point in z direction. The circle positions represent the vortex positions and the diameter represent their vorticity. The velocity profiles are the velocities on the center lines in perpendicular direction.

determined here can be used for the transient test cases as well. It also reflects the procedure of running a simplified and coarse precursor simulation to verify the simulation setup and determine a good configuration, as would be done in preparation of long-running simulations.

Parameter	Value
H	$0.41m$
L	$2.5m$
D	$0.1m$
l_x	$0.45m$
l_y	$0.15m$
(x_a, y_a, z_a)	$(0.45m, 0.20m, 0.205m)$
(x_e, y_e, z_e)	$(0.55m, 0.20m, 0.205m)$

Table 5.9 Geometry of the square cylinder and channel.

The overall setup of the flow around a square cylinder is shown in Figure 5.12 and the corresponding geometry specification is listed in Table 5.9. This setup, along with the following characteristic numbers, is also used in the transient examples in Sections 5.4.1 and 5.4.2. The channel walls parallel to the flow direction as well as the cylinder are no-slip boundaries. The

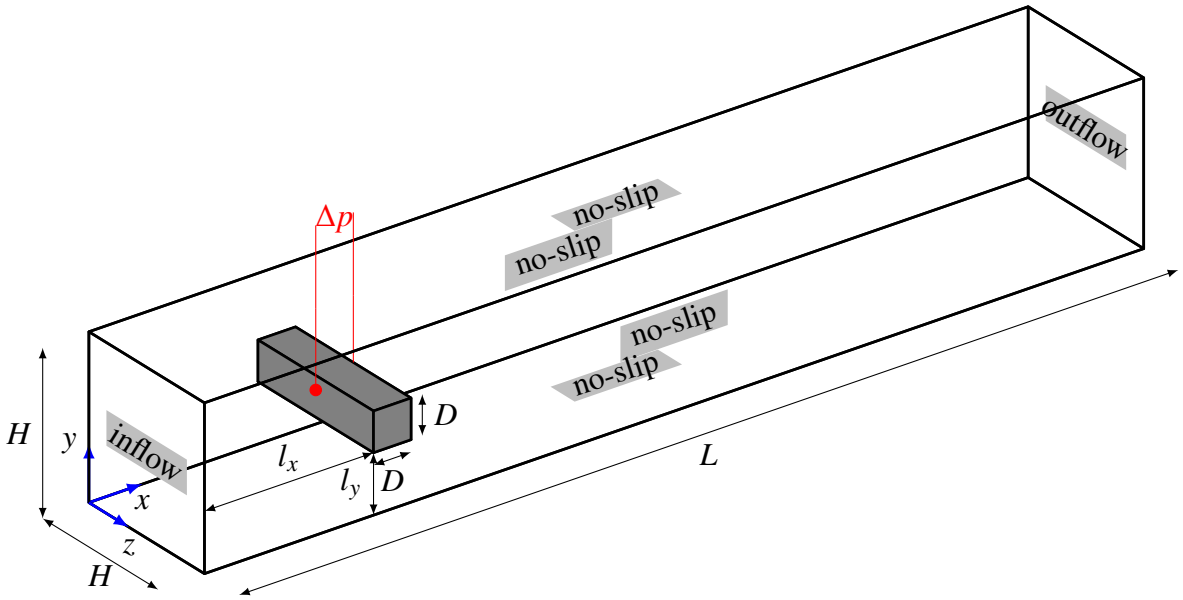


Figure 5.12 Setup of the flow around a square cylinder. Adapted from [158].

inflow is realized as a parabolic profile prescribing the velocities according to

$$\boldsymbol{U}(0, y, z) = \begin{pmatrix} U_m \frac{16}{H^4} yz\,(H - y)\,(H - z) \\ 0 \\ 0 \end{pmatrix} \tag{5.9}$$

in dependence of the maximum inflow velocity U_m. The maximum inflow velocity is related to the characteristic velocity $\overline{U}$ as $\overline{U} = \frac{4}{9} U_m$ and the Reynolds number is defined by $\mathrm{Re} = \frac{\overline{U} D}{\nu}$ where ν is the kinematic viscosity. The channel outflow at $x = 2.5m$ is treated in the same manner as for the duct flow introduced in Section 5.3.1 and has a pressure of $p = 0$ prescribed.

The quantities of interest used to evaluate the performance are the non-dimensional lift coefficient c_L, the drag coefficient c_D and the pressure difference ΔP, defined as

$$c_L = \frac{2F_L}{\rho \overline{U}^2 DH} \tag{5.10}$$

$$c_D = \frac{2F_D}{\rho \overline{U}^2 DH} \tag{5.11}$$

$$\Delta P = P(x_a, y_a, z_a) - P(x_e, y_e, z_e). \tag{5.12}$$

The computation of lift and drag forces F_L and F_D, respectively, are method-specific and are given in detail later. Similarly the pressure on the cylinder surface is not available directly for the lattice-Boltzmann method and its recovery is provided in Section 5.3.3.4.

The parameters specific to the configuration of the laminar flow around the square cylinder are listed in Table 5.10. The Mach and Reynolds numbers as well as the density ρ and velocity U_m determine the flow. The derived parameters are also given for convenience. Reference values for the quantities of interest are taken from [129].

Parameter	Value
Ma	0.1
Re	20
ρ	$1\ \frac{\mathrm{kg}}{\mathrm{m}^3}$
U_m	$0.45\ \frac{\mathrm{m}}{\mathrm{s}}$
$\overline{U}$	$0.2\ \frac{\mathrm{m}}{\mathrm{s}}$
ν	$0.001\ \frac{\mathrm{m}^2}{\mathrm{s}}$

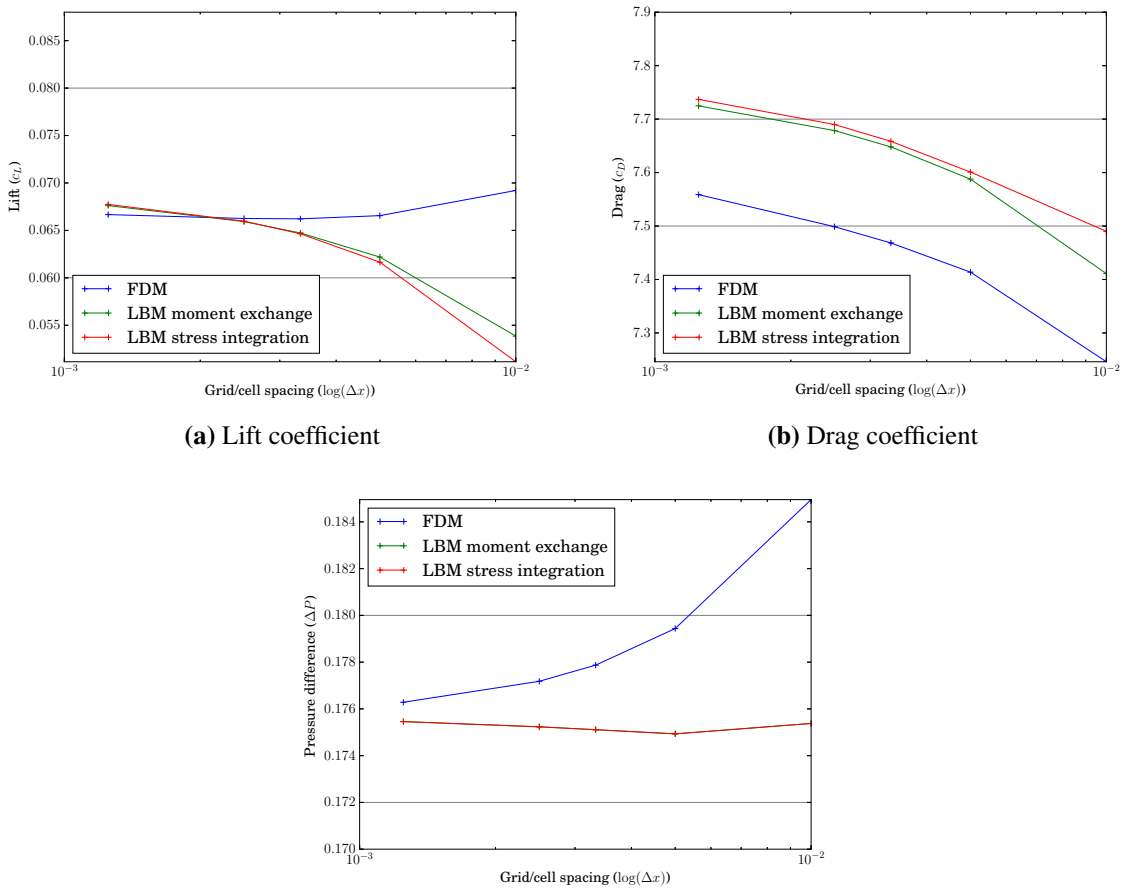
Table 5.10 Parameters specific to the laminar flow around a cylinder with a square cross section.

5.3.3.2 Comparison considerations

In the following considerations for the comparison that are common to both methods are given. In the previous test cases all tuning parameter could be chosen freely. However, for this case the possible discretization resolutions are severely limited. Due to the slightly asymmetric channel geometry all grids must be able to evenly divide a length of 0.01 m. Considering the high number of simulations that are required in the optimization process the three coarsest possible resolutions are chosen for the comparison. This limitation is imposed as the parameter tuning becomes exceedingly expensive for high resolutions and, more importantly, the reference solution used for the error evaluation must be sufficiently finer resolved and still be at a feasible cost. The corresponding grid spacings are 0.01, 0.005 and $0.00\bar{3}$, which is equivalent to 442764, 3451389, and 11361879 FD points or 465948, 3542112, and 11750000 LB cells. These large increments make the process of optimizing towards a specific error and comparing the TTS impossible. Instead the process is reversed, which means the TTS for a range of target errors is measured at a fixed resolution. The interpolated TTS are then compared between between both methods.

As for the LDC a reference solution has to be established. Each method again utilizes its own simulations on a very fine mesh, in order not to discriminate either scheme due to differing convergence behavior. The flow past a square cylinder is known for its sensitivity and with these two fundamentally different approaches a matching solution cannot be expected. The fully converged steady-state solutions for different resolutions shown in Figure 5.13 highlight these deviations. The techniques to evaluate lift, drag and so forth are given in detail in the method-specific Sections 5.3.3.3 and 5.3.3.4. The diagrams also show the lower and upper bound for the respective benchmark quantities established in [129]. Note that the drag for the LBM lies outside this range, which matches the findings in [129], where an LBM based solver also obtained a too high drag coefficient of $c_D = 8.093$. As these bounds were determined simply by picking the results that were considered trustworthy, they cannot be taken as absolute. This wide variation in possibly valid results is why each method is timed against its own result from the finest grid. As the number of possible discretizations is limited and the number of target errors is important for accurately interpolating the TTS it chosen larger than before. The target error is evaluated for 24 logarithmic steps between 1 and 10^{-3} and is calculated as a relative error to ease the comparison of different quantities of interest.

The tuning is again performed with an extensive search for optimal simulation parameters and code configuration. In order to approach a process that more closely relates to practical



(a) Lift coefficient

(b) Drag coefficient

(c) Pressure difference

Figure 5.13 Spatial convergence of the lift and drag coefficient and pressure difference for the steady-state flow past the cylinder with a square cross section. The lift and drag results for the LBM have been recovered with two different techniques. The horizontal gray lines show the range of valid results as determined in [129]. Adapted from [158].

engineering work, only one set of simulation parameters will be used for all resolutions. Due to the effort involved, tuning per every specific simulation shows only benefits when many instances of that configuration will be run. In many cases however, the slightly longer simulation runtime will be an acceptable penalty for the reduced effort. The code parameters on the other hand, are tuned for every resolution level. The idea behind this is that automatic tuning can easily be applied without requiring user interaction. Alternatively the parameter can be tabulated according to the hardware platform and domain dimensions, which makes it easy to retrieve the optimal set.

5.3.3.3 Finite difference method

As noted earlier, the lift, drag and pressure difference calculations are specific to each method. The pressure difference is straight forward to determine as the pressure on the cylinder surface is readily available. Depending on the number of points with which the cylinder surface is discretized, however, the FD point may not coincide with the evaluation points (x_a, y_a, z_a) and (x_e, y_e, z_e). In that case a cubic interpolation is employed to avoid adverse effects on the convergence order. The lift and drag forces are obtained by evaluating the integrals

$$F_D = \int_S \left(\rho\nu \frac{\partial u_t}{\partial n} n_y - p n_x \right) dS, \tag{5.13}$$

$$F_L = \int_S \left(\rho\nu \frac{\partial u_t}{\partial n} n_x - p n_y \right) dS \tag{5.14}$$

on the cylinder surface S and its normal $\boldsymbol{n} = (n_x, n_y, n_z)^T$, where u_t are the tangential velocities. In order to evaluate these integrals in a finite difference context the differentials are substituted by single-sided third order differences and the four sides are evaluated as weighted sums analogously to equation (5.8).

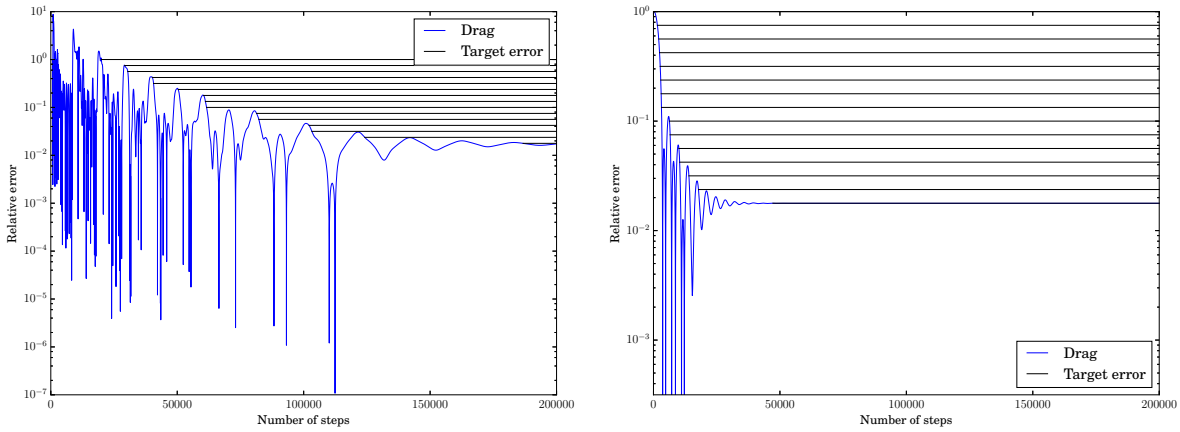
The reference result are computed with the lift and drag evaluation routines detailed above and the parameters in Table 5.11. The small grid spacing results in a very high resolution of 81 points along the short edge of the cylinder. Due to the use of globally uniform grids it leads to a significant memory requirement of approximately 20 GiB, even with the generally low memory footprint of an FDM with rectangular grid in a low-storage time integration. Due to the CFL

Parameter	N_x	$N_y = N_z$	Δx	ρ	ν	Ma	c_{velo}	c_{pres}	CFL
Value	2001	329	0.00125	1	0.001	0.1	0.1	0.8	0.1
QOI	c_L	c_D	Δp						
Result	0.06667	7.55874	0.17628						

Table 5.11 Parameters and results for the FDM reference simulation.

number restriction a total of 690000 steps are required for convergence with a runtime of 162 hours, starting from an extrapolated solution of a simulation with a grid one level coarser.

Through the available reference the required steps per target error can be evaluated. The development of the error over time is shown exemplary with the medium grid spacing of 0.005 in Figure 5.14a for the drag coefficient. It clearly exhibits the oscillatory behavior expected from artificial compressibility methods. A target tolerance is therefore considered to be reached when



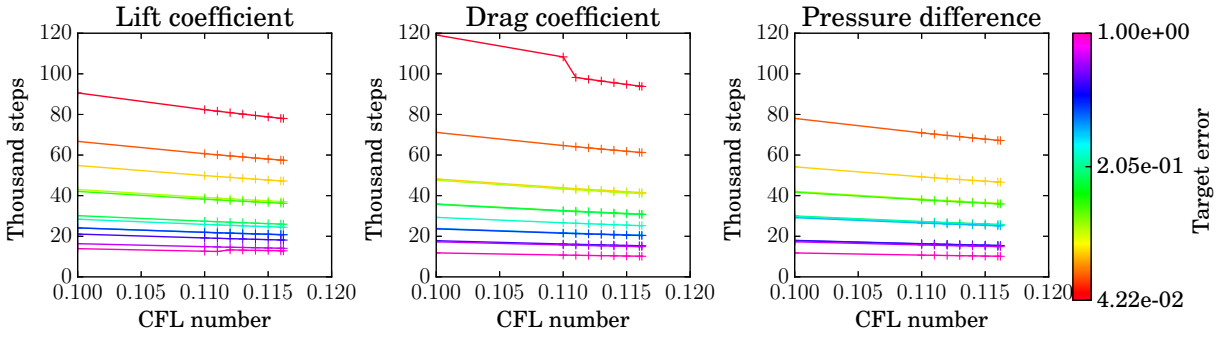
(a) FDM **(b)** LBM

Figure 5.14 Relative error of the drag coefficient over the number of steps and the achieved target error using the fully tuned simulations at a grid spacing of 0.005.

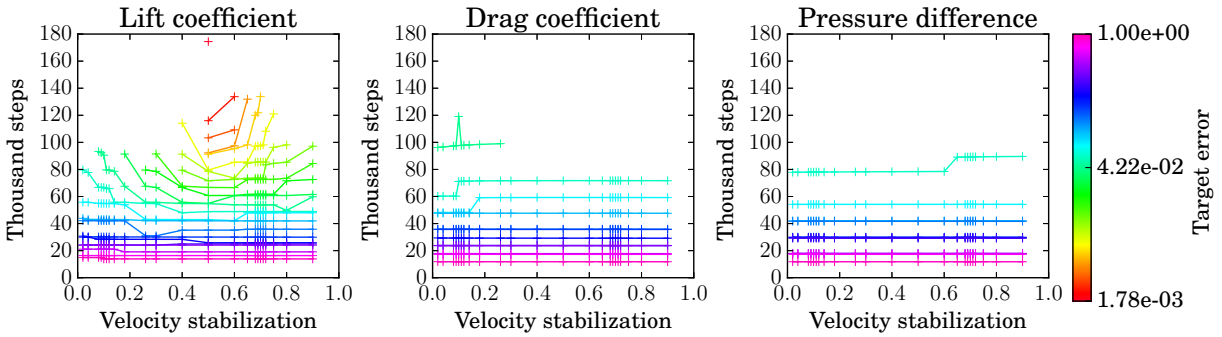
the error will not increase above that level again. This way the resolution and the target tolerance can be linked to a specific number of steps and by extension to a certain walltime.

In order to achieve the highest performance possible it is essential to minimize the required number of steps. This was previously achieved by extensive searches for optimal parameters for every resolution, target error and characteristic measure. This process is simplified for this test case in order to more closely resemble a practical engineering procedure and is introduced in the following. The parameters available for tuning are the CFL number, c_{velo}, c_{pres} and the number of steps for ramping up the inflow. With regard to the transient test cases of the flow around a cylinder the ramping is not applied. The effect of ramping was minor for the previous tests and in transient simulations the temporal evolution of the inflow profile is prescribed anyways. Figure 5.15 shows the influence of the remaining three parameters on the number of steps for the three characteristic measures. The behavior varies widely by either parameter and measure. Only the tendency for the CFL number in Figure 5.15a is clear, as the number of steps decreases for an increasing CFL number for all QOI until the stability limit is reached. For the velocity and pressure stabilization in Figure 5.15b and 5.15c however opposing trends can be seen for the different characteristic measures. This leads to the question of which measure is most relevant for the performance. Considering the engineering background it is the largest error that determines the overall simulation quality. As the target errors are evaluated at fixed intervals, it means that the maximum number of steps to reach the target tolerance of either QOI has to be taken into account. Figure 5.16a depicts this maximum in steps for the same results as before and which forms the first step of parameter tuning.

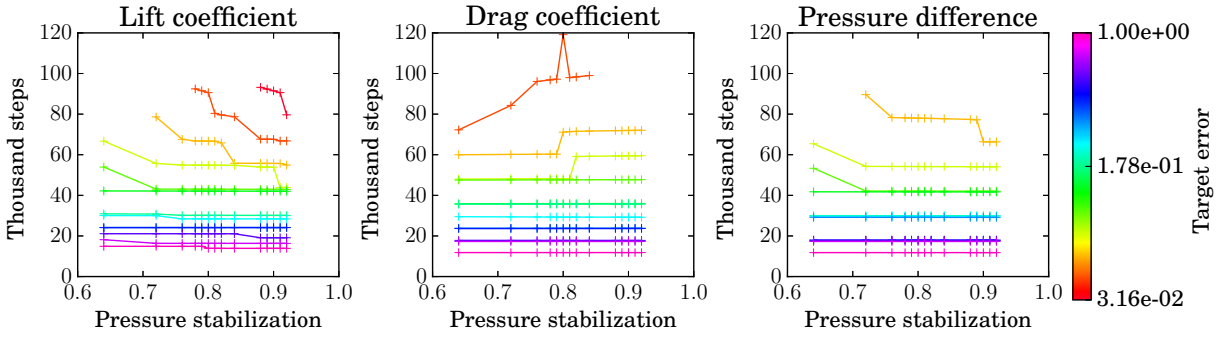
To determine the absolute best parameter combination the entire space of combinations has to be searched. This is infeasible and better search strategies have to be used. As before the combination of an integer output and strong fluctuations in the results from the oscillatory convergence behavior makes automated optimization exceedingly difficult. At the same time these oscillations allow the performance gains to be separated into two categories. The category offering the large gains is when a parameter change causes an entire oscillation cycle to be skipped (see Figure 5.14a for an exemplary convergence behavior). The other category of performance



(a) CFL number



(b) Velocity stabilization

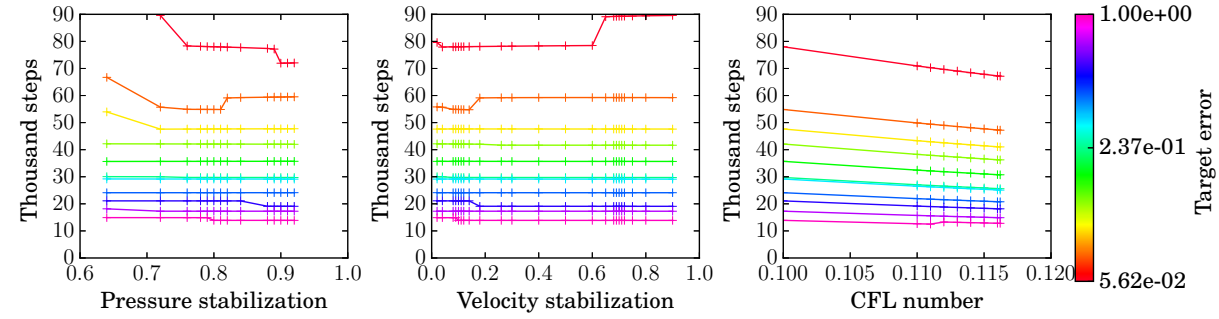


(c) Pressure stabilization

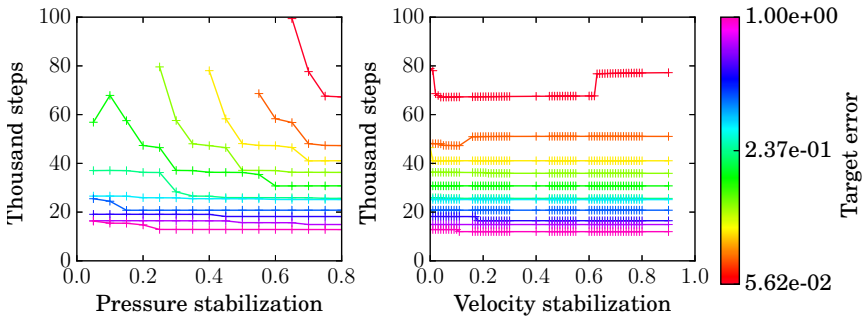
Figure 5.15 Influence of simulation parameters on the number of steps required by the FDM to reach the target tolerance. All parameters are held at their initial values of CFL = 0.1, $c_{pres} = 0.8$ and $c_{velo} = 0.1$ apart from the parameter under investigation. The colors represent the different target tolerances.

gains merely lead to a slight shift in the aforementioned oscillations and typically only reduces the total number of steps by single digit numbers. Considering that a single step makes up for 0.0004% to 0.01% percent in the total runtime this is within the noise introduced by the time measurements. It is therefore important to achieve the best result within the first category of performance gains. An additional limiting factor to the usefulness of marginal gains stems from the single set of parameters for all resolutions. This by itself causes some deviation from the ideal configuration. However, in the following it can be seen that these deviations are small and are unlikely to have a significant impact on the methods' overall performance.

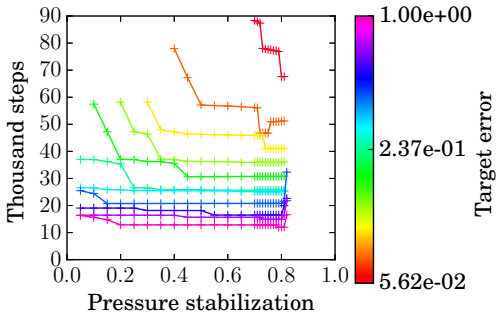
The simulation parameter tuning is carried out one parameter at a time. This sequential approach reduces the computational effort by orders of magnitude. The different stages of the parameter selection are shown in Figure 5.16. In the first stage all three tunable parameters



(a) First stage



(b) Second stage



(c) Third stage

Figure 5.16 Maximum number of steps required to reach the relative target error for lift, drag and pressure difference by the FDM. In each stage only the shown parameter was varied. The used grid spacing is 0.01.

are varied across the full potential range and the most relevant ranges are investigated in detail. These variations are carried out while retaining the initial values for all other parameters. The most promising of the parameters is then selected and is set to its optimal configuration for the following stages. Here the CFL number is chosen for the first stage due to its clear tendency for all observed values and tolerances. For the second stage, shown in Figure 5.16b, the same process is repeated for the remaining parameters and with the CFL number at its optimal configuration of 0.116. As the velocity stabilization is mostly agnostic to change it is used next. The particular value of $c_{velo} = 0.54$ was chosen for the tighter lift tolerances that can be achieved and the reduced number of steps in reaching them (see Figure 5.15b). For the third stage only the pressure stabilization parameter is left and it is selected as 0.74, which is a compromise that offers good results for most of the tolerances.

As mentioned previously the goal is to find a set of parameters that works well for all used resolutions. From a practical standpoint the most exhaustive search should be performed on the coarsest grid and then be verified for the finer grids. In Appendix A.3 the same process as before is provided, but with a grid spacing of 0.005. It is evident that the parameters chosen before work equally well on this grid. The final parameters are summarized in Table 5.12.

CFL number	c_{velo}	c_{pres}
0.116	0.54	0.74

Table 5.12 The optimized simulation parameters for the flow around a cylinder with a square cross section using the FDM.

Apart from the simulation parameters there is a tuning opportunity with the code parameters. These are almost independent of the actual simulation and the optimal configuration is purely a function of the hardware and the domain dimensions. As noted in Section 5.3.3.2 these are optimized for every grid size as the configuration could in practice be read from predetermined lookup tables or through automated tuning. The results are presented in Table 5.13.

Resolution	block size x	block size y	time per step [s]
0.01	128	3	0.00322
0.005	512	12	0.02720
$0.00\bar{3}$	760	6	0.08259
0.0025	504	5	0.23220
0.002	628	10	0.46924
$0.001\bar{6}$	756	8	0.75217
$0.00\overline{142857}$	588	9	1.41823
0.00125	504	10	2.05433

Table 5.13 Optimized FDM code parameter for the flow around cylinder with a square cross section domains.

5.3.3.4 Lattice-Boltzmann method

The challenges with simulating the flow around the cylinder with a square cross section using the lattice-Boltzmann method are slightly different than the ones for the FDM. There are no

simulation parameters left for optimization as the permissible cell sizes are determined by the channel geometry and the inflow ramping is not used with regard to the transient test cases that follow.

Instead the challenge lies in evaluating lift, drag and pressure difference in the first place. Two different approaches can be used, which are the stress integration and momentum exchange [91, 169]. The stress integration technique is essentially identical to the method used for the FDM, as the equations (5.13) and (5.14) are evaluated using the velocity recovered through equations (3.4) and the pressure with equation (3.11). The use of the half-way bounce back boundary condition leads to some additional complications as the quantities of interest are not available on the cylinder surface. For the velocity gradient there is an easy solution, as the velocity on the boundary is by definition zero and the differences used must merely be adjusted to the half step distance. The pressure is more cumbersome to calculate as it has to be extrapolated from the domain onto the cylinder surface. A cubic extrapolation using the nearest three points normal to the surface is used for this evaluation. The same interpolation strategy on the cylinder surface as for the FDM is used in addition to the pressure extrapolation, when the center points of the cylinder sides do not coincide with cell locations.

The alternative and more accurate approach for the evaluation of the lift and drag forces is the momentum exchange technique [169]. It utilizes the fact that the LBM is based on densities moving at specific speeds and which thus carry a momentum. When these densities reach the BB condition they are reflected back and the transferred momentum can be evaluated. The force on the entire boundary is computed by

$$\boldsymbol{F} = \sum_{\boldsymbol{x}_b} \sum_{i_b(\boldsymbol{x}_b)} 2 f_{i_b}(\boldsymbol{x}_b, t)\, \boldsymbol{c}_{i_b} \frac{\delta_x^4}{\delta_t^2}, \tag{5.15}$$

where $\boldsymbol{x}_b$ are cells with density distribution functions entering the obstacle and $i_b(\boldsymbol{x}_b)$ are the corresponding directions leaving the domain. The difference in the results for stress integration and momentum exchange is comparatively small as seen in Figure 5.13. Since the momentum exchange yields results that are slightly closer to the reference it is used exclusively from here on. However, the pressure difference must still be evaluated in the same fashion as for the stress integration version.

Provided with these techniques the reference results are evaluated, with the parameters and the results in Table 5.14. The simulation required 350000 steps and 419 hours until convergence,

Parameter	Δx	ω	QOI	C_L	C_D	Δp
Value	0.00125	1.23775	Result	0.06761	7.72475	0.17546

Table 5.14 LBM reference parameter and results for the steady-state flow past a cylinder with a square cross section.

starting from a quiescent field. Due to the 19 densities per point the memory consumption of 60.5 GiB is over three times higher than of the FDM simulation with an equivalent grid spacing.

Figure 5.14b shows an exemplary convergence of the drag coefficient with a relative error to the above reference solution. As expected the solution converges in a similar oscillatory manner as the FDM. However, there are some important differences between the two. The LBM exhibits much less high frequency oscillation than the FDM. Furthermore the LBM requires significantly

fewer steps to converge, which, assuming a comparable time per step, should mean an advantage for the LBM. However, there is the drawback of an overall larger error, which thus requires a higher resolution than for the FDM.

As noted previously there are no simulation parameters available for optimization. However, there are code parameters that are subject to tuning. Table 5.15 shows the results of an extensive search for optimal parameters for each used cell spacing. These parameters are reused for the

Cell spacing	memory block size	index block size	time per step [s]
0.01	254	1	0.00313
0.005	212	1	0.01862
$0.00\bar{3}$	252	1	0.05972
0.0025	356	1	0.14063
0.002	212	1	0.30085
$0.001\bar{6}$	218	1	0.37325
$0.00\overline{142857}$	220	1	0.74253
0.00125	252	1	1.10513

Table 5.15 Optimized code parameter for the LBM.

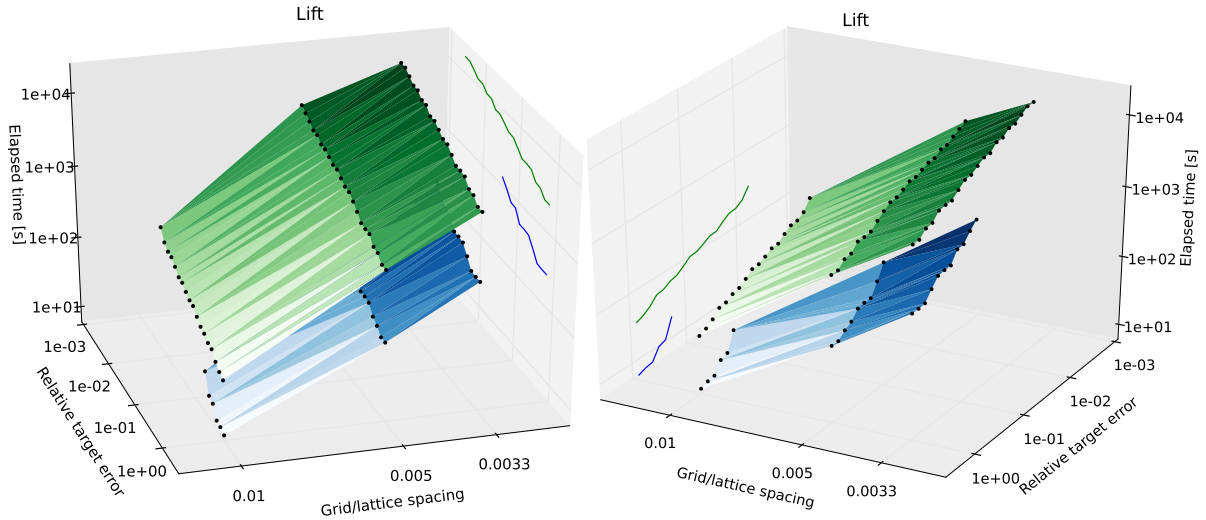
transient test cases in Section 5.4.1 and 5.4.2, as also for the LBM the code tuning only depends on the domain dimensions and these are identical for all flow around a cylinder examples.

5.3.3.5 Results

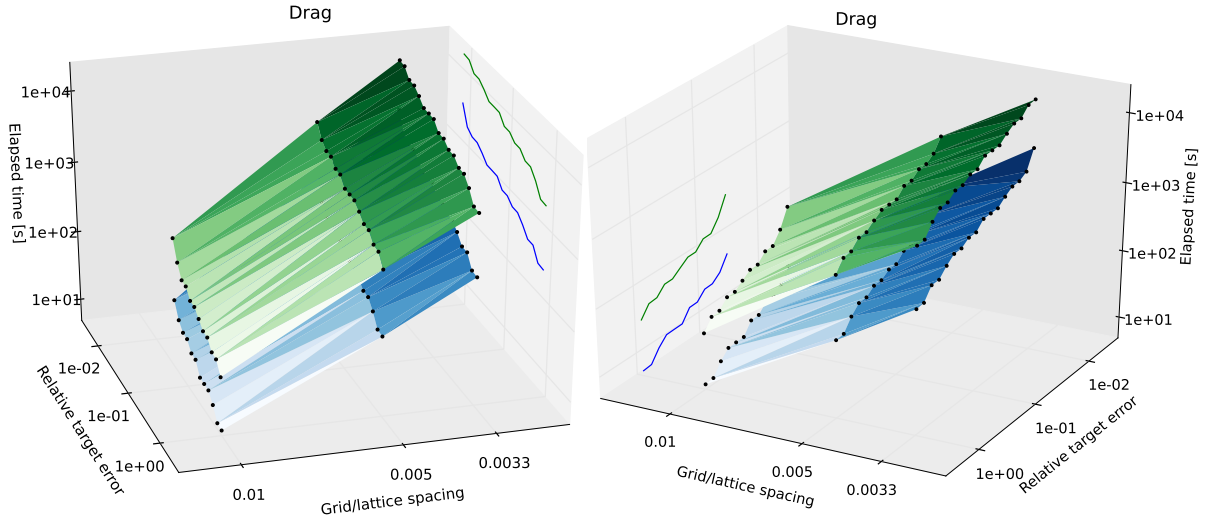
With the full set of optimal simulation and code parameters determined according to the above procedure, the methods can be compared on their performance in terms of runtime. However, there are some important differences to the previous comparisons performed on the duct flow and lid driven cavity. From those examples it is clear that the highest performance is reached at the lowest domain resolution that is still capable of reaching the target tolerance. In the case of the flow around a cylinder this flexibility of a free resolution choice is not given, instead the geometry dictates large increments in the point distance or cell size. As this prohibits using the ideal configuration that ensures the best performance, a different comparison strategy than comparing the TTS at specific tolerances has to be used.

In order to gain an overview of the overall performance of both methods, Figure 5.17 lists the time to solution for the full set of solutions. It shows the TTS depending on the resolution and target error. It can clearly be seen that for a given resolution and target error the LBM is significantly faster, regardless of the quantity of interest. As the planes in the logarithmic representation are nearly parallel, the LBM is always faster for this comparison at all feasible resolutions. In order to quantify this difference Table 5.16 shows the timing differences for the strictest tolerance that both methods are able reach simultaneously. This performance figure has a very limited use as it does not compare the true performance of each method. The choice of the grid spacing for comparison is quite arbitrary and using the number of degrees of freedom for example would completely alter the results.

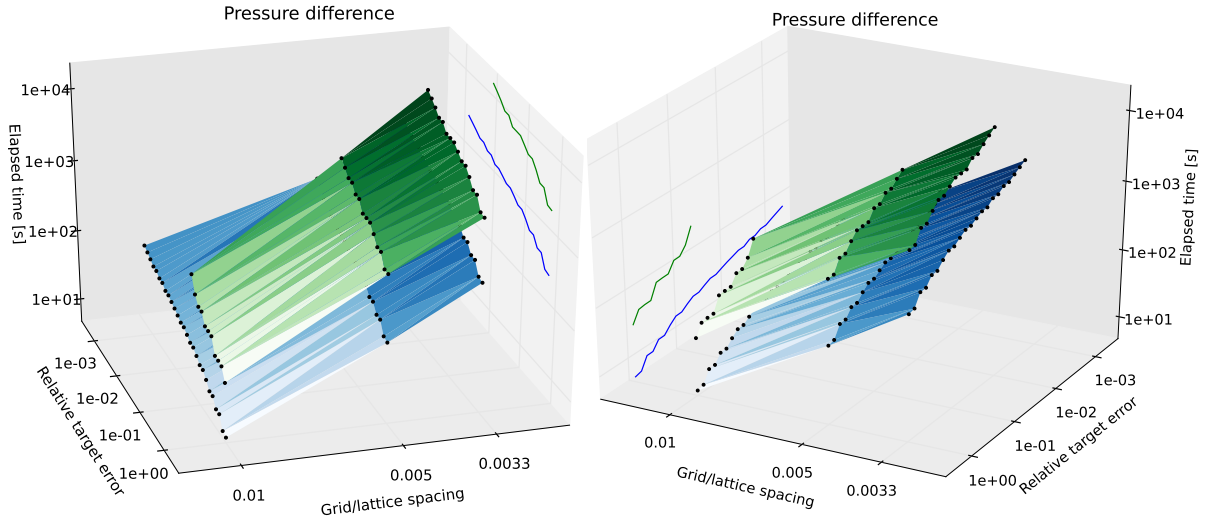
Both approaches exhibit their optimal performance at the minimal resolution per target error, which cannot be chosen freely. Hence the process is reversed and the strictest possible tolerance is



(a) Lift



(b) Drag



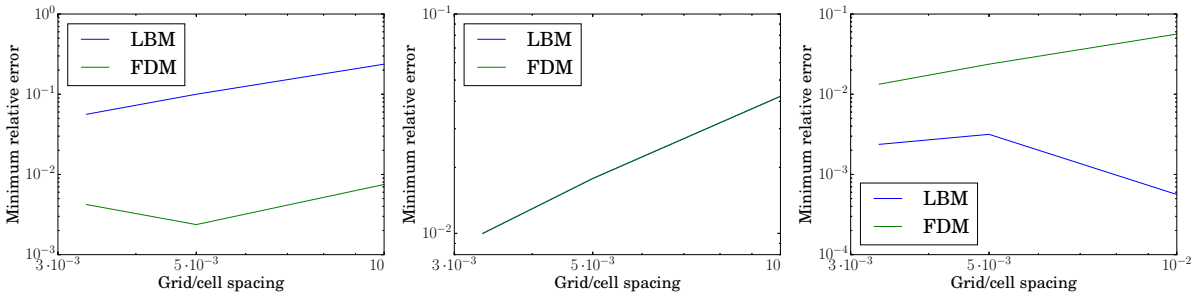
(c) Pressure difference

Figure 5.17 Total time to solution depending on the target error and resolution with all axis shown logarithmic. Shown are the results for lift, drag and pressure difference across the cylinder for the LBM (blue) and FDM (green).

QOI	Spacing	Target error	Duration FDM	Duration LBM	Timing ratio
Lift	0.01	0.237 14	78.74	23.64	3.33
	0.005	0.1	1783.05	347.25	5.13
	$0.00\bar{3}$	0.056 23	9059.34	1696.78	5.34
Drag	0.01	0.042 17	364.96	47.47	7.69
	0.005	0.017 78	5374.69	970.09	5.54
	$0.00\bar{3}$	0.01	20 904.7	4326.34	4.83
Pressure difference	0.01	0.056 23	250.81	23.20	10.81
	0.005	0.023 71	3864.33	345.19	11.19
	$0.00\bar{3}$	0.013 34	18 938.4	1978.41	9.57

Table 5.16 Comparison of the total time to solution between the FDM and LBM for all tested resolution and at the lowest common target error.

evaluated for the three available resolutions available per method. These are shown in Figure 5.18. Naturally the obtained tolerances per method do not match up, which inhibits a direct comparison and other strategies are developed in the following. Figure 5.18 gives a very mixed picture, as



(a) Lift **(b)** Drag **(c)** Pressure difference

Figure 5.18 Minimum reached error per resolution and method for different quantities of interest.

the FDM clearly reaches lower tolerances for the lift, while it is the opposite for the pressure difference. For the drag both methods can reach identical error levels. This is likely caused by the challenge that the pressure field before and after the cylinder poses to the FDM in the employed configuration. Different stabilization technique could possibly remedy this situation, but are outside the scope of this work. The quality of the pressure result is of course essential for the pressure difference and still plays an important part in the drag. In contrast and as demonstrated by the performance figures the lift is not affected by these pressure irregularities.

The points given in Figure 5.18 represent the points at which each method reaches their optimal performance. Apart from the drag these points do not match and cannot be compared directly. However, the performance ratio can be approximated. For this the smallest achievable error of the code with the overall worse error is used as a reference to which the other method is matched. This is done by determining the coarsest resolution, which is able to reproduce that error. This way the timings that are compared are at the optimal configuration for the first code and at a lower bound of performance for the second. Therefore two distinct cases can arise. If the first code is

faster than the second, the established difference is the maximum performance difference, i.e., the first implementation is at most that much faster than the second implementation. If however the second code is determined to be faster, then it is at least that much faster, possibly more. Table 5.17 shows the result of this procedure for the three tested quantities.

QOI	Method	Spacing	Target error	Duration [s]	Timing ratio
Lift	LBM	$0.00\bar{3}$	0.05623	1696.78	12.7
	FDM	0.01		133.28	
Drag	LBM	$0.00\bar{3}$	0.01	4326.34	0.207
	FDM	$0.00\bar{3}$		20904.7	
Pressure difference	FDM	$0.00\bar{3}$	0.01334	18938.4	586.4
	LBM	0.01		32.299	

Table 5.17 Performance comparison of the optimal configuration of the method with the least small error (upper rows) with the coarsest solution with matching error of the other method (lower rows). The timing ratios are minima as the second/matched method could be faster if different spatial resolutions could be used.

As before there is no definite answer to the question of which is the faster approach, but there are still some conclusions that can be drawn. The overall trend shows that the LBM converges with fewer time steps than the FDM and due to the very comparable walltime per time step it is faster if tolerance and resolution are prescribed. Spatial convergence however exhibits a different behavior. It was shown that the FDM requires much lower resolution in space than the LBM for the same relative error in lift, while it was the opposite for the pressure difference. This is also reflected in the results for the lower bound of the performance difference. When obtaining the same relative error of 0.056 in lift the FDM was at least 12.7 times faster and for an relative error of 0.013 in the pressure difference the LBM was at least 586 times faster. For the drag error comparison both methods used the optimal configuration and the LBM was faster by a factor of 5 for the target error of 0.01. This shows that the performance depends drastically on the considered quantity of interest and either method can be faster. The performance snapshot shown here does lead to the conclusion that both methods are competitive as neither fully outperformed the other on all QOI.

The situation is even more difficult if the results are to be compared against the ones in [129]. These are only given as the final values with few significant digits and exhibit a very wide spread as shown by the bounds in Figure 5.13, which means that the result quality cannot be quantified properly. Simply selecting a successful FDM and LBM simulation and comparing the duration would lead to an arbitrary outcome. Furthermore, the published results are for simulations with significantly fewer degrees of freedom up to 6.7 million unknowns and much slower hardware at a LINPACK performance up to 1408 MINSTR/s than used in this work. The simulations carried out in this work with 46 million unknowns on a platform with a LINPACK performance of 565.7 GINSTR/s is much faster in absolute numbers, but also inherently less efficient per hardware performance.

5.4 Time-dependent flows

5.4.1 Fully developed unsteady flow around a cylinder with a square cross section

5.4.1.1 Problem description

The test case with unsteady flow is an extension to the flow around a cylinder with a square cross section introduced in Section 5.3.3. This configuration is referred to as 3D-Q2 in [129], which again provides some reference results for the comparison applied here. This example of laminar unsteady flow has also been used for comparison of the LBM with the finite volume method [13], though not with regard to performance.

The FDM and LBM under consideration are weakly compressible flow solvers, which means they are not time-accurate for unsteady problems. This should be taken into account when configuring unsteady simulations and an appropriate configuration, most importantly in terms of the Ma number, has to be found to mitigate the compressibility effects. It is then possible to solve unsteady problems with an acceptable accuracy. This matches the goals of this work as not the most accurate, but fast and sufficiently accurate solvers are investigated. For the artificial compressibility method used with the FDM, there are even techniques to solve fully incompressible flows time-accurate. These dual-timestepping techniques employ a pseudo-time for the temporal derivative in the continuity equation to achieve a truly incompressible steady-state within every time step [105, 152]. However, as no comparable approach is available for the LBM this option is not pursued further as it would skew the comparison.

The geometry remains the one shown in Figure 5.12 and the boundary conditions, evaluation routines and quantities of interest also remain largely unchanged. Merely the inflow velocity U_m in the inflow profile given in (5.9) is increased. The velocity is chosen such that an unsteady

Parameter	Value
Re	100
ρ	$1\ \frac{\mathrm{kg}}{\mathrm{m}^3}$
U_m	$2.25\ \frac{\mathrm{m}}{\mathrm{s}}$
$\overline{U}$	$1\ \frac{\mathrm{m}}{\mathrm{s}}$
ν	$0.001\ \frac{\mathrm{m}^2}{\mathrm{s}}$

Table 5.18 Parameters specific to the unsteady flow around a cylinder with a square cross section.

flow in form of a von Kármán vortex street develops behind the cylinder. The corresponding parameters are summarized in Table 5.18. This test case is limited to the fully developed flow and the ramp up to the fully developed state is not considered. This type of transient behavior is investigated in the next test case in Section 5.4.2.

The same quantities of interest as for the steady-state flow past a cylinder in the previous section are observed here. However, the lift coefficient in equation (5.10), the drag coefficient in equation (5.11) and the pressure difference in equation (5.12) are now time dependent. As an

additional measure the Strouhal number

$$\mathrm{St} = \frac{Df}{\overline{U}} \tag{5.16}$$

is introduced. It is a non-dimensional measure for the vortex shedding frequency f which allows for an easy comparison with experimental and simulation results. It is also very relevant for engineering application as the induced oscillation must often be addressed. The frequency f is defined through the oscillations of the lift.

5.4.1.2 Comparison considerations

The transient nature makes a practical comparison even more difficult than in the previous examples. In order to give a direct ratio on the performance a single number must describe the solution quality which can be set in relation with the time. With more than one quantity of interest several performance figures can be created and compared independently. For the unsteady flow however drag, lift and pressure difference are functions of time and cannot be correlated that easily. These quantities are therefore analyzed qualitatively. For this three full oscillations of the fully developed flow, aligned to the c_L oscillation, are examined. An exception is the Strouhal number which describes a property of the developed flow through a scalar and is used for a quantitative comparison.

The benchmark computations in [129] are used as reference values. However, these results vary on such a wide scale that no lower or upper bound is given. Taking the St number for example, it varies between 0.2777 and 0.3488 for the highest resolution versions alone. In one case the flow even reaches steady-state, which incidentally occurred for an LBM based code and which will be important later on. In [129] the maximum lift and drag, as well as the St number, are used for comparison. As noted in that reference this gives very little insight into the unsteady flow pattern. These reference numbers give a rough estimate on the anticipated order of magnitude, but cannot be used for quantifying the performance. Here the average over three oscillations and their amplitude are used instead. This characterizes the sinusoidal behavior of the quantities of interest much better.

Apart from the performance analysis this test case also serves in establishing the envelope in which the codes can be applied sensibly. Most quantities are only compared qualitatively, however they show some significant artifacts when approaching each method's limits. Both approaches are analyzed in this regard for spatial resolution and the prescribed Ma number. As previously the spatial resolution is restricted to very specific spacings due to the channel geometry. The used spacings are 0.01, 0.005 and $0.00\bar{3}$ and additionally a cell size of 0.0025 for the LBM. The preferred Ma number for either method is around 0.1, but the alternatives 0.2 and 0.05 are tested as well.

With regard to code tuning the parameters from the steady-state flow around a cylinder with a square cross section can be reused. The geometries are identical, and since the flow pattern does not influence the execution speed in terms of time per step for an explicit solver these values remain optimal. This also underlines the strategy of optimizing the code parameters for every resolution as a practical approach. This is unlike the simulation parameters which do depend on the simulation setup. However, in order to reduce the effort and more closely follow engineering practices, the simulation parameters here are based on the previous test case, which is taken as a precursor simulation.

5.4.1.3 Finite difference method

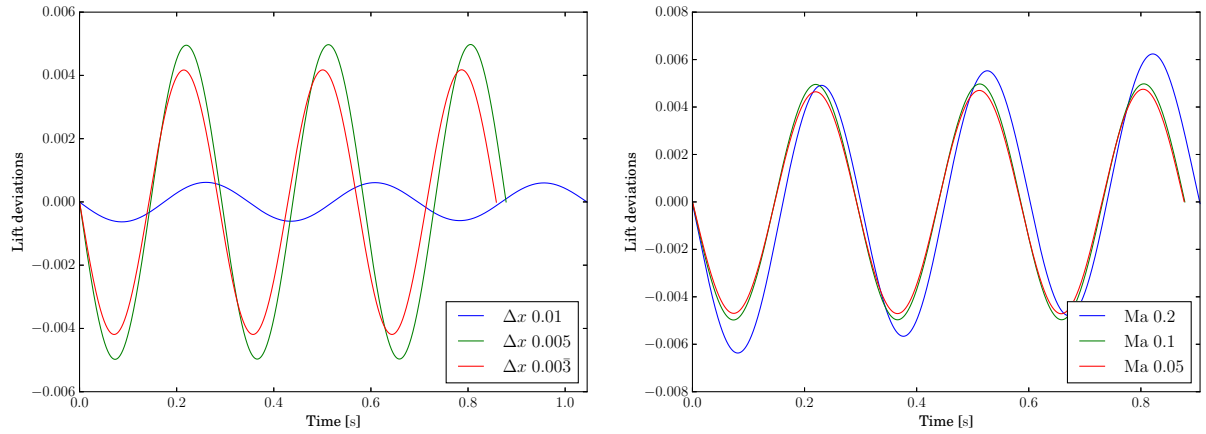
The employed FDM is based on the artificial compressibility approach, which can lead to some difficulties when time-accuracy is needed. However, small compressibility effects can usually be tolerated in many practical applications. As mentioned before, there are dual-timestepping techniques to circumvent the compressibility, but they are not applied here due to a lack of an LBM equivalent.

The lack of accurate reference values requires that the influence of the spatial resolution and Ma number on the results is analyzed in detail to ensure that the method is used in a reliable regime. It also serves the purpose of uncovering the limitations within which a sensible operation is possible. For all analysis three periods of the lift oscillation in the fully developed flow are used. They are shown in Figure 5.19 for the three quantities of interest. The corresponding numbers on averages and amplitudes can be found in Table 5.19 alongside the St number and the required number of steps for three periods. Considering the spatial resolution a significant difference can be seen between the point distance of 0.01 and the finer resolutions. The St number is considerably lower and the amplitude is significantly smaller. This effect is very pronounced in Figure 5.19a. In Figures 5.19b and 5.19c showing the drag and pressure difference another important difference emerges. The frequency with which the drag and pressure difference oscillate is not double the lift oscillation frequency, which is roughly the case with the higher resolution. This indicates that the 0.01 grid spacing is not sufficient to capture important flow patterns and is thus outside the usable FDM configuration. The other spatial resolutions match sufficiently well and the lift, while it is far from converged, shows a clear trend. A similar effect can be observed for the drag and pressure difference.

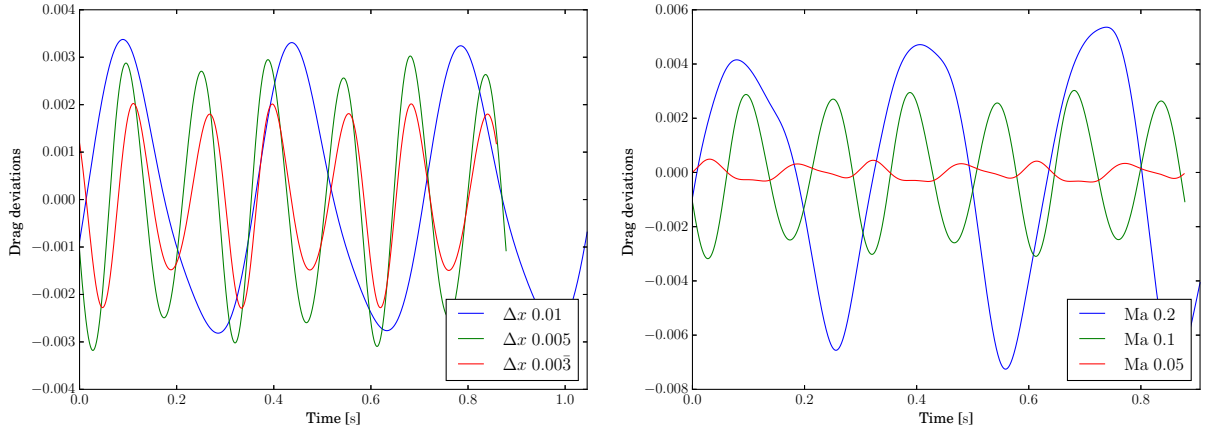
The influence of the Ma number is more distinctive. With increasing Ma number the amplitude of the oscillations in all quantities of interest increases. This is not surprising as a flow at a higher Ma number is less “stiff” against oscillations, leading to larger amplitudes and lower frequencies. Interestingly the drag and pressure difference at Ma = 0.2 exhibit approximately half the frequency than for Ma = 0.1. With lower Ma numbers like Ma = 0.05 the oscillations are irregular, likely due to the superposition of multiple effects like acoustic modes that persist over a longer time span. A Ma number close to 0.1 should therefore be maintained to obtain sensible results.

For the flow at Ma = 0.2 there is a further effect which is not obvious from Figure 5.19, but which can be observed clearly when more than three periods are shown as in Figure 5.20a. The expected oscillation in the lift from the vortex shedding is superimposed with a much larger and lower frequency oscillation. As it does not decrease over time it must be driven through the flow and is not an artifact of the initial ramping. This effect was only observed for Ma = 0.2, making this Ma number unsuitable for the FDM used here.

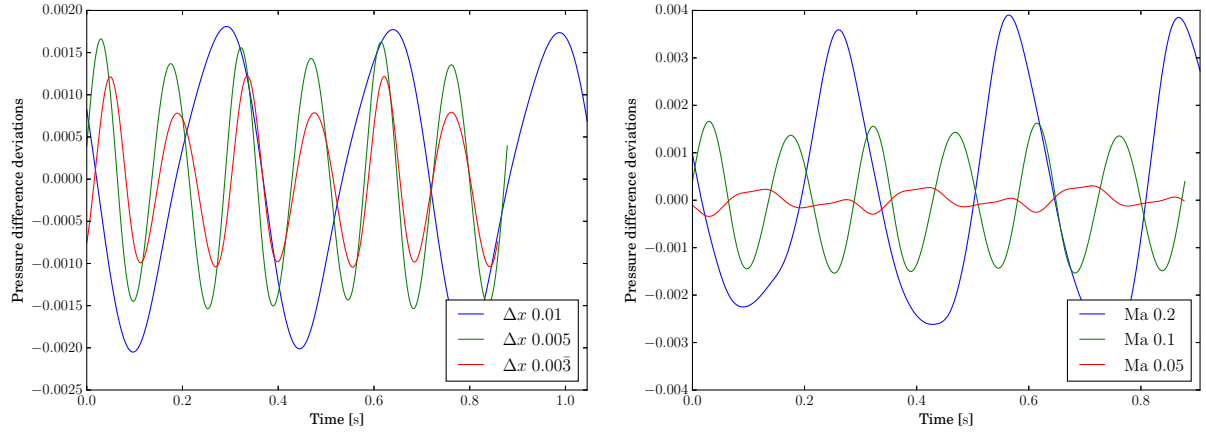
Returning to the operation envelope of the artificial compressibility FDM some conclusion can be drawn from the above findings. The resolution must be at least at a grid spacing of 0.005 or else the flow pattern changes completely. An even higher resolution is advisable, as some quantities still show significant changes at resolution modifications. For the Ma number it is clear that Ma = 0.1 offers the best results, which matches the design point of the presented ACM. Higher Mach numbers lead to superfluous fluctuations and lower Mach numbers increase the number of time steps and leads to high frequency oscillations, which takes excessively long time to dissipate. Furthermore, the development of vortex shedding is hampered.



(a) Lift

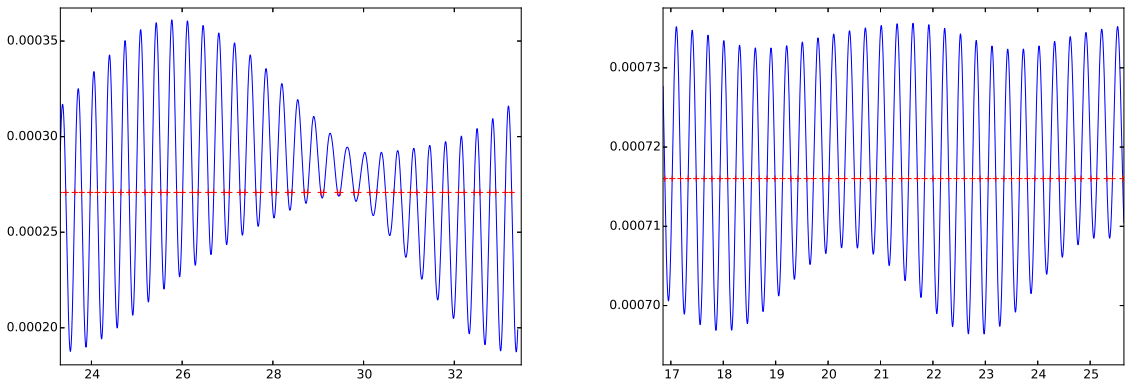


(b) Drag



(c) Pressure difference

Figure 5.19 Three periods of oscillation, simulated with the FDM at different grid resolutions at Ma = 0.1 and different Ma numbers at a grid spacing of 0.005. The different measures are aligned according to the lift oscillations.



(a) FDM at Ma = 0.2 and point distance of 0.01

(b) LBM at Ma = 0.2 and cell size of 0.0025

Figure 5.20 Lift oscillation due to vortex shedding superposed with longer time scale oscillation only observed at the higher Mach number (Ma = 0.2) and hence higher compressibility.

For a comparison to the LBM the QOI presented above are at most suited for a qualitative description, but not for an accurate performance comparison. This is especially problematic with both methods at such a similar performance level. However, the Strouhal number promises to be a useful option as it describes the unsteady behavior in a single quantity and the results from the FDM simulation are sensible and within the range found in [129]. Unfortunately no reliable reference value for the St number is available that would allow a convergence study to be performed and a reference simulation using the FDM itself at a sufficiently high resolution takes an unfeasible long time to reach the fully developed state. Additionally the same approach would have to be pursued for the LBM, which is difficult as detailed in Section 5.4.1.4.

As mentioned earlier the code parameters do not change for different flow patterns of the flow past a cylinder as the domain dimensions remain identical. The specific configuration is listed in Table 5.13 in the previous test case. The simulation parameters on the other hand are affected by the increased flow velocity and unsteady behavior. The strategy is to reuse the configuration from the previous example as a precursor simulation. These parameters, listed in Table 5.12, work well also in the new conditions. This has been verified with attempts at a further performance increase, which lead to instabilities for the CFL number and no significant change for the pressure and velocity stabilization.

5.4.1.4 Lattice-Boltzmann method

An analogous investigation on the influence of resolution and Mach number as for the FDM in Section 5.4.1.3 is performed for the LBM. The QOI are again evaluated for three periods of the fully developed flow. Figure 5.21 shows the influence of different resolutions at Ma = 0.1 and different Mach numbers at a cell size of $0.00\bar{3}$ on the lift, drag and pressure difference. All successful simulations are summarized in Table 5.20 in terms of the average over the three periods and their amplitude. Additionally the Strouhal number and number of steps are provided.

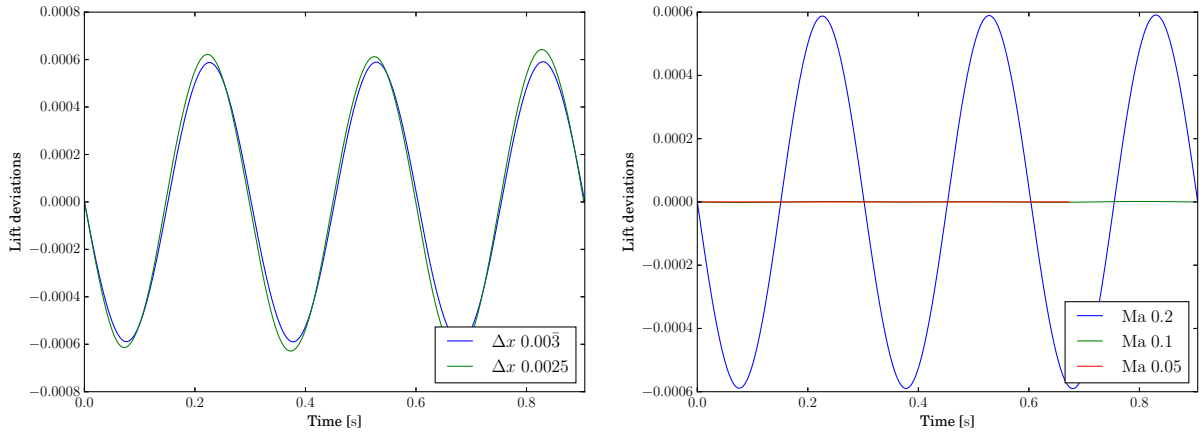
As can be seen from Table 5.20 and Figure 5.21 no results were obtained for the cell sizes of 0.01 and 0.005. This is due to instabilities that arise at the inflow with the higher inflow velocity, compared to the steady-state case. The LBM simulation is non-dimensionalized such that the cell

Grid spacing	Ma	St	avg(c_L)	amplitude(c_L)	avg(c_D)	amplitude(c_D)	avg(Δp)	amplitude(Δp)	Number of steps
0.01	0.2	0.2880	0.012 23	0.006 25	4.961 74	0.016 37	3.996 87	0.010 35	12 519
	0.1	0.2870	0.016 74	0.001 25	5.044 61	0.006 19	4.082 57	0.003 86	22 717
	0.05	0.2928	0.019 80	0.002 27	5.166 24	0.002 03	4.208 85	0.002 42	42 156
0.005	0.2	0.3313	0.014 77	0.012 61	4.833 76	0.012 64	3.812 48	0.006 99	21 884
	0.1	0.3415	0.017 79	0.009 95	4.903 42	0.006 20	3.841 70	0.003 20	38 285
	0.05	0.3420	0.018 92	0.009 46	4.992 74	0.000 84	3.891 70	0.000 64	72 268
$0.00\bar{3}$	0.2	0.3476	0.021 38	0.010 28	4.729 66	0.011 01	3.743 11	0.005 76	31 348
	0.1	0.3495	0.021 67	0.008 37	4.783 71	0.004 31	3.756 10	0.002 27	56 160
	0.05	-	-	-	-	-	-	-	-

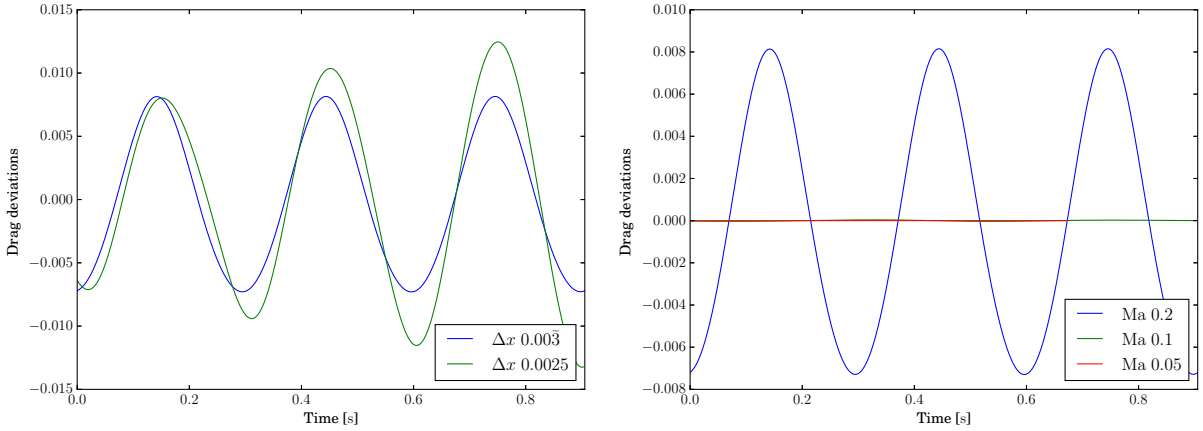
Table 5.19 Full set of results for the unsteady flow around a cylinder with a square cross section simulated with the FDM at multiple resolutions and Mach numbers.

Cell size	Ma	St	avg(c_L)	amplitude(c_L)	avg(c_D)	amplitude(c_D)	avg(Δp)	amplitude(Δp)	Number of steps
0.01		-	-	-	-	-	-	-	-
0.005		-	-	-	-	-	-	-	-
$0.00\bar{3}$	0.2	0.331 36	0.033 83	0.001 18	4.215 81	0.015 45	3.573 21	0.009 00	5294
	0.1	0.333 74	0.033 98	$3.40 \cdot 10^{-6}$	4.215 67	$7.33 \cdot 10^{-5}$	3.573 09	$4.75 \cdot 10^{-5}$	10 511
	0.05	0.445 78	0.033 98	$2.51 \cdot 10^{-8}$	4.215 67	$3.58 \cdot 10^{-6}$	3.573 09	$2.00 \cdot 10^{-6}$	15 738
0.0025	0.2	0.334 29	0.034 88	0.001 83	4.292 31	0.014 40	3.570 61	0.008 73	6997
	0.1	0.212 05	0.035 37	$4.07 \cdot 10^{-5}$	4.291 75	0.002 10	3.570 23	0.001 37	22 056
	0.05	0.448 91	0.035 38	$2.37 \cdot 10^{-6}$	4.291 71	0.000 32	3.570 21	0.000 18	20 838

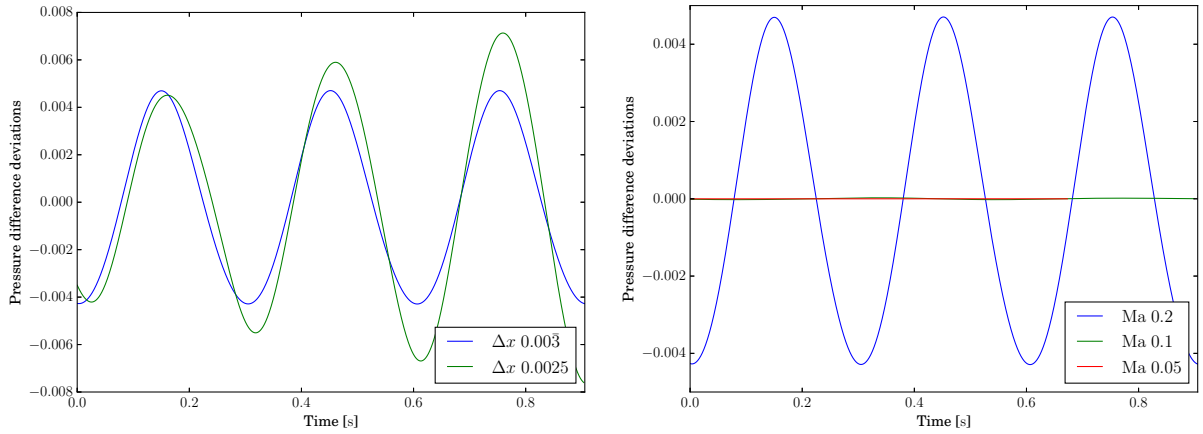
Table 5.20 Full set of results for the unsteady flow around a cylinder with a square cross section simulated with the LBM at multiple resolutions and Mach numbers.



(a) Lift



(b) Drag



(c) Pressure difference

Figure 5.21 Three periods of oscillation simulated with the LBM at different cell sizes at Ma = 0.2 and different Ma numbers at with a cell size of 0.00$\bar{3}$. The different measures are aligned according to the lift oscillations.

size, time step size and therefore velocity are always one in lattice-Boltzmann units. Hence there is no CFL criterion that indicates an unstable configuration. However, the relaxation parameter is directly influenced by the characteristic values used for non-dimensionalization and must be within specific limits to retain a stable method [164]. Apart from the relaxation parameter the Mach number must remain small as well, since only special variations of the LBM can cope with transsonic flow. Here however, neither are outside the stable region and in fact the instabilities do not arise at the cells with the highest velocity, but at the inflow. The very simple inflow condition has more stringent requirements towards the maximum inflow velocity. This is no particular problem however, as higher resolutions work flawlessly and since the LBM requires significantly fewer steps for simulating the three oscillations it can easily make up for the higher resolution.

Figure 5.21 also shows the lack of oscillation for $Ma < 0.2$. While these oscillations can still be determined computationally, Table 5.20 shows that the amplitudes are multiple orders of magnitude smaller than at $Ma = 0.2$. This indicates that the LBM exhibits increased viscosity leading to a smaller effective Reynolds number, dropping back into the steady regime. This increased numerical viscosity is underlined by the increased drag detected in the previous steady-state example, highlighted in Figure 5.13b. Simulations at $Re = 120$ overcome the additional viscosity and the von Kármán vortex street develops for $Ma = 0.1$ as well. With a $Ma = 0.2$ an unsteady flow can be achieved at $Re = 100$ despite the additional dissipation. This is caused by the higher compressibility, less "stiff" flow and therefore higher amplitudes of oscillations, observed not only for the LBM, but the FDM as well. This lets the flow at $Ma = 0.2$ overcome the higher viscosity and vortex shedding is recovered, as the transition occurs for a Mach number between 0.18 and 0.19. However, this configuration is not without problems either as can be seen from low frequency oscillation superposed on the expected lift oscillation shown in Figure 5.20b. Nonetheless it is the best option for a comparison available here.

In terms of simulation parameter optimization there are no free parameters available. The number of ramping steps is irrelevant as only the fully developed flow is considered and the spatial resolution is fully tested within the stability and practicable simulation duration range. The code parameters have been tabularized for the steady-state case in Table 5.15 and can be used unaltered.

5.4.1.5 Results

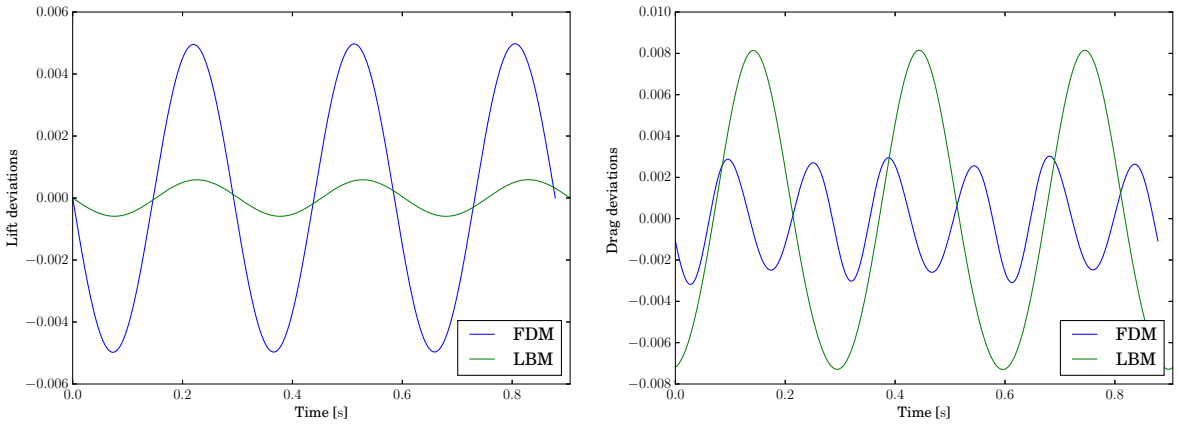
From the above findings for the FDM and LBM simulation it has to be concluded that a direct quantifiable comparison of performance is not possible for the unsteady flow around a cylinder with a square cross section within the scope of this work. All quantities that have been analyzed are not suitable in some fashion. This is partly due to the difficulty of comparing result quality, in particular without reliable references, and in other parts the difficulty in finding parameters for which both methods yield the expected results. Especially the conflict in the Mach number selection, as the LBM exhibits the desired vortex shedding only for flows at large Mach numbers while the FDM works around $Ma = 0.1$.

A promising candidate was the Strouhal number, but as it is strongly influenced by the Mach number achieving a desired Strouhal number becomes an exercise in adjusting the Mach number rather than establishing a methods accuracy. Exactly the same holds true for the amplitude of the lift, drag and pressure difference oscillation. This leaves the averaged quantities, however, they do not contain any information on the unsteady nature of the flow and give very similar results

independent of whether the flow has developed a full vortex street or not. This can be clearly seen in the results of the LBM simulation at Ma = 0.2 and the almost steady flow at Ma = 0.1.

A further difficulty is the lack of a definite reference. The flow around a square cylinder is very challenging as it is very sensitive. Checking convergence towards a solution published in literature is not a suitable comparison, due to the different solutions the methods converge to. This is underlined by the range of possible results that are published in [129] and which can at most be used to verify the overall agreement. This is the case for the results of the FDM. The maximum lift is in the middle of the given range while the maximum drag falls into the range for higher resolutions as well. The St number is also well inside the plausible range, but this is even the case for the simulations at the coarsest resolution deemed not suitable. This calls the usefulness for comparison of that particular measure into question. The LBM simulations at Ma = 0.2 agrees with the reference values, but obviously not at lower Mach numbers. Interestingly the reference data includes an LBM method, denoted as number six in [129], for which no Strouhal number is provided. This could indicate that it also remained in a steady-state.

To give at least a rough qualitative comparison the fastest acceptable solution from either approach are analyzed. For the FDM this corresponds to the configuration at Ma = 0.1 and with an grid spacing of $0.00\bar{3}$ as the most accurate solution and the LBM is used with Ma = 0.2 as only then the flow turns unsteady and in conjunction with a cell size of 0.0025 for the most accurate result. In Figure 5.22 these two configurations are compared with regard to lift and drag



(a) Lift **(b)** Drag

Figure 5.22 Comparison of the best tested configurations of FDM and LBM simulations of the unsteady flow around a cylinder with a square cross section.

coefficients. It is apparent that there are significant discrepancies and without a definite reference the better solution cannot be determined. However, there are some noteworthy features, as for example the LBM exhibits a smaller amplitude in the lift oscillation despite being additionally excited to develop the vortex street through the use of a higher Mach number. Also the frequency in the drag oscillation of the LBM is approximately half that of the FDM. The combination of these two effects strongly resembles the dismissed FDM solution at a point spacing of 0.01. Possibly the excitation through the higher compressibility leads to a different oscillation mode or perhaps one of the simulations is still underresolved for this problem.

Despite the apparently not comparable solutions an impression on the simulation runtimes is given in Table 5.21. For each method two exemplary versions are presented. The data shows

Method	Grid spacing	Ma number	Time for three periods [s]
FDM	0.005	0.1	1101.7
	$0.00\bar{3}$	0.1	4634.9
LBM	$0.00\bar{3}$	0.2	316.1
	0.0025	0.2	2134.9

Table 5.21 Performance figures for select configurations in simulating the three periods of the unsteady flow past a cylinder with a square cross section.

that the LBM clearly is faster on a purely resolution based comparison due to the significantly larger allowable time step size. This however gives no indication on the actual performance in regards to solution quality. In fact the findings indicate that the LBM has difficulties in recovering the von Kármán vortex street in the first place. The FDM on the other hand does recover the expected behavior, in particular for higher resolutions and at suitable Mach numbers. This means of course that the FDM is clearly the preferable approach for this exact test case independent of an actual runtimes. In the following section a similar test case is evaluated that is less sensitive and therefore less challenging, which avoids most of the problems encountered here.

5.4.2 Transient flow around a cylinder with a square cross section

5.4.2.1 Problem description

The third version of the flow around a cylinder is a transient test case with a time dependent inflow condition, requiring time-accurate solutions. It is based on the setup labeled “3D-3Q” in [129]. The simulated time goes from 0 s to 8 s with an inflow profile prescribed according to

$$\boldsymbol{U}(0, y, z, t) = \begin{pmatrix} U_m \frac{16}{H^4} yz\,(H-y)\,(H-z) \sin\left(\pi \frac{t}{8}\right) \\ 0 \\ 0 \end{pmatrix} \quad \forall\, t = [0, 8], \tag{5.17}$$

which is the same spatial profile as before, but with a superposed temporal sinus function. All further simulation parameters are given in Table 5.22. The boundary conditions, choice of discretizations and the evaluation of the quantities of interest remain identical to the preceding unsteady test case.

The weakly compressible behavior of both methods poses an even bigger challenge than for the fully developed unsteady flow. This is due to the lack of a settling phase between simulation start and the error evaluation in which acoustic modes can be dissipated and where some of the compressible effects due to transients balances out. These difficulties are amplified by the sudden increase in inflow velocity due to non-smooth ramping at the beginning of the sine curve.

The quantities used to asses the performance are the development of the lift and drag coefficients. The pressure difference is no longer considered as it offers no additional insight over the drag. Also the Strouhal number does not apply here and different scalar measures have to be found for a quantitative performance assessment.

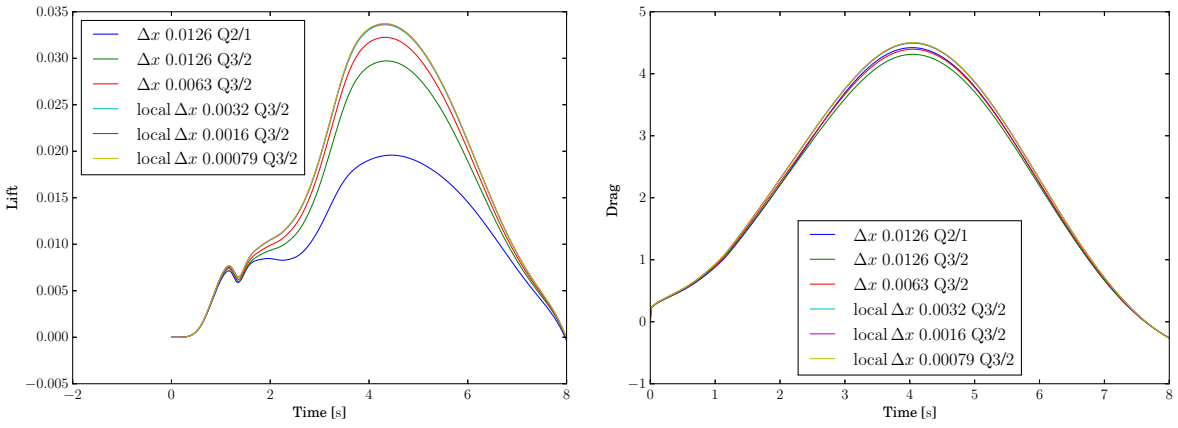
Parameter	Value
Re	100
ρ	$1\ \frac{\mathrm{kg}}{\mathrm{m}^3}$
U_m	$2.25\ \frac{\mathrm{m}}{\mathrm{s}}$
$\overline{U}$	$1\ \frac{\mathrm{m}}{\mathrm{s}}$
ν	$0.001\ \frac{\mathrm{m}^2}{\mathrm{s}}$

Table 5.22 Parameters specific to the transient flow around a cylinder with a square cross section.

5.4.2.2 Comparison considerations

The comparison is similarly challenging to the previous case of unsteady flow as the quality of the solution has to be evaluated over the entire time span. Then there is also the specific problem of the weakly compressible nature, which leads to oscillation and other artifacts being overlaid with the actual solution and which are particularly hard to quantify.

In order to give insight into the accuracy and performance the QOI are analyzed in several different ways to give an overall picture of the performance. The temporal development of both methods is shown for a variety of configurations, which is used to confirm the findings on the most suitable setup determined in the previous test case. This is done individually for each method. These examinations are carried out with the help a truly incompressible solution. This reference solution is obtained with the implicit higher order finite element solver Adaflo [89], which also provided a reference solution to the lid driven cavity in Section 5.3.2.1. The results for different spatial resolutions and element orders is given in Figure 5.23. The solution with the highest resolution is converged well beyond what the FDM and LBM will reach in the following and is used as the reference. However, these results are not only used as a reference solution,



(a) Lift **(b)** Drag

Figure 5.23 Reference solution generated with implicit incompressible flow solver Adaflow.

but also to give an impression on the resource consumption in comparison between implicit and explicit solvers in the later results section.

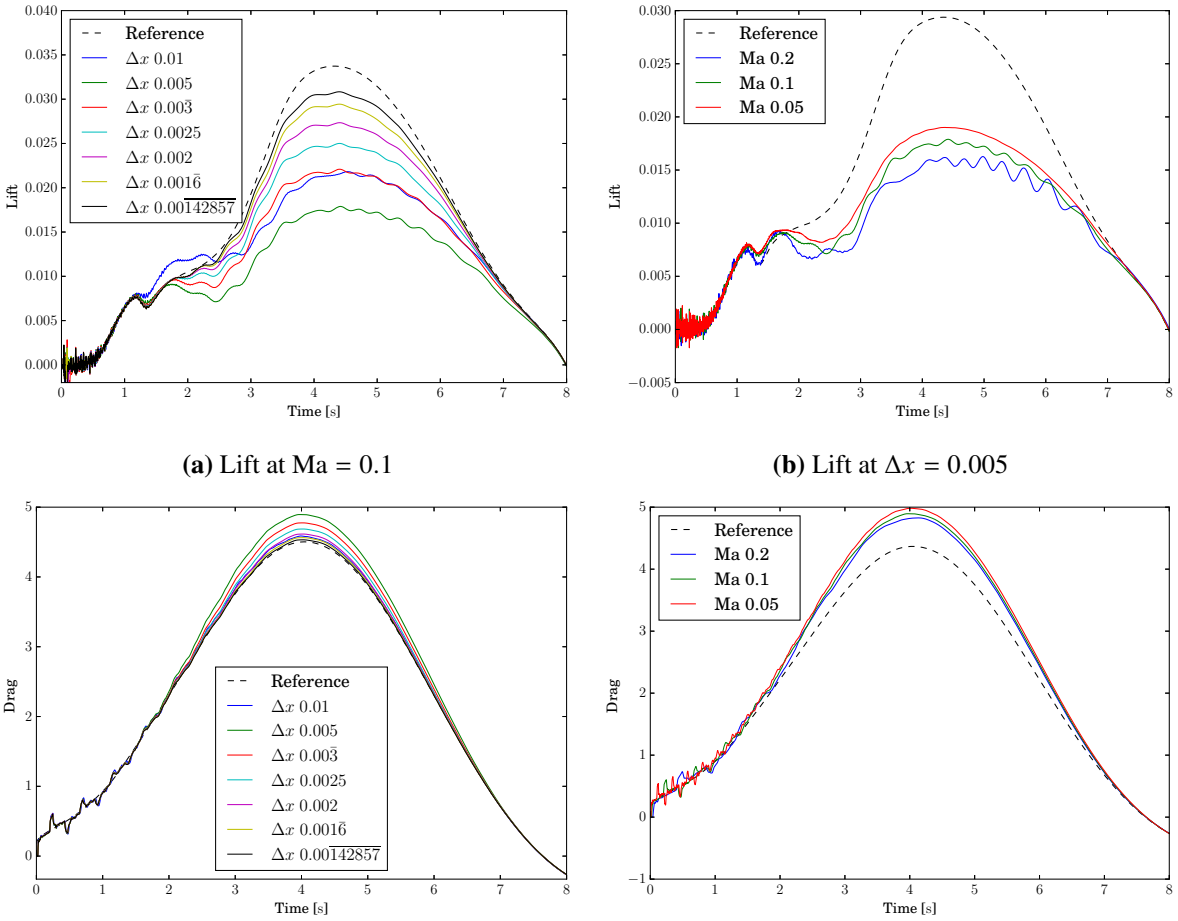
Apart from the qualitative comparison of the time dependent behavior several different strategies for a quantitative comparison are analyzed. An integral measure over the entire time span would be possible, however it will be discussed within the results section why the obtained simulation results hinder its use. As an alternative several different characteristic properties of the results are investigated. These are scalar and can therefore be used to directly evaluate the performance. However, many of them also suffer from similar difficulties as the integral error evaluation.

On top of the intricacies of applying a fair and comprehensive error measure the problem with the limited spatial resolution still hinders the absolute comparison of both method's performance. The comparison process is therefore reversed by comparing solutions with a very similar runtime against each another. This shows which solution can be expected in which runtime. Naturally these are only snapshots and do not give an all-encompassing picture over all possible configurations. However, the range of resolutions that are tested include every possible increment from the coarsest to the finest resolution that can sensibly be simulated on the reference hardware. The Mach number is also varied between the 0.05, 0.1 and 0.2 configuration.

5.4.2.3 Finite difference method

Figure 5.24 shows the influence of the resolution and the Mach number on the result. Apart from the coarsest grid at $\Delta x = 0.01$, which was established as insufficient for the unsteady flow already, a clear convergence towards the reference solution can be seen. However, unlike the truly incompressible reference solution additional oscillations are superposed. These occur on two different scales. The high frequency oscillations that vanish after 2.3 s are the result of the non-smooth start, which causes a shock to travel back and forth through the channel and slowly disperse. The lower frequency oscillations that persists throughout almost the entire simulation have multiple causes. On the one hand the compressible nature of the ACM causes all flow changes to be slightly delayed and fluctuate by a small degree, which naturally also applies to the ramp-up and down of the inflow velocity. On the other hand it is the onset of vortex shedding. As shown for the fully developed unsteady flow in the previous Section 5.4.1 the Re number at the point of the maximum inflow velocity is high enough to induce a vortex street. However, due to the short period in that regime the flow cannot transition completely. The higher Ma number used in 5.24b makes the effect more pronounced, just as it was the case for the fully developed unsteady flow. This makes Ma = 0.1 a sensible choice for the comparison, as the number of steps grows almost reciprocal to the Mach number and at Ma = 0.1 and with higher resolutions the spurious oscillation are comparatively small.

As for the previous case the tuned code parameters are kept identical to the ones from Section 5.3.3.3 as the optimal configuration depends on the hardware and domain dimensions alone. Similarly the same simulation parameters as with the previous test cases are maintained. The runtime measurements can therefore be executed with the identical configuration as before, apart from the inflow velocity. The results of these runtime measurements can be found in Table 5.23, together with the LBM timings for direct comparison.



(a) Lift at Ma = 0.1

(b) Lift at $\Delta x = 0.005$

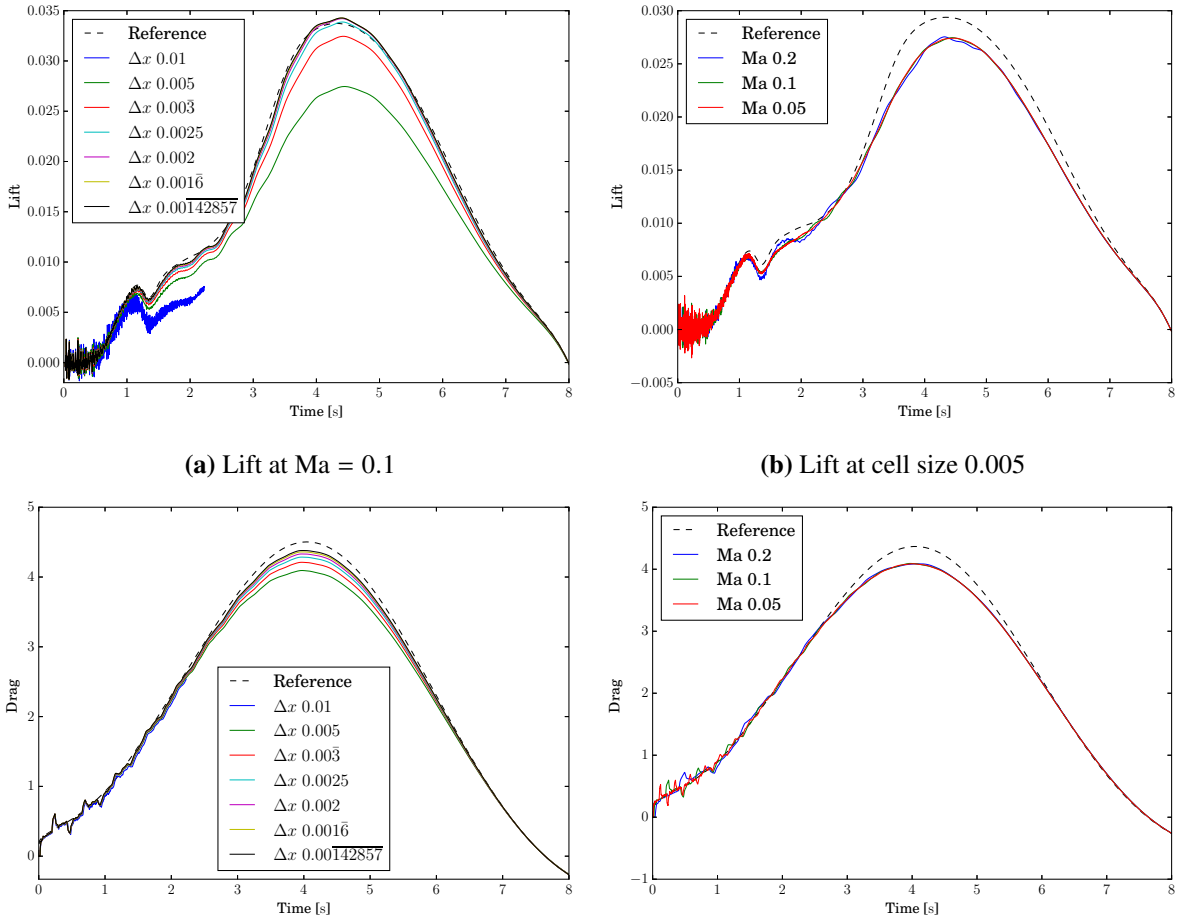
(c) Drag at Ma = 0.1

(d) Drag at $\Delta x = 0.005$

Figure 5.24 Lift and drag coefficient simulated with the FDM over the time span of 8 s for different grid spacings at Ma = 0.1 and for different Mach numbers at a point distance of $\Delta x = 0.005$. Adapted from [158].

5.4.2.4 Lattice-Boltzmann method

The same configurations as with the FDM above are tested with the LBM and it exhibits many similarities. However, the coarsest spatial resolution develops the same stability problems at the inflow as for the unsteady example. The lift and drag coefficient results for various resolutions are shown in Figure 5.25 and there is a clear convergence towards a common result. For the lift coefficient this solution lies past the reference solution. Hence the LBM converges towards a different result and which makes a fair comparison difficult when assessing the result quality. On



(a) Lift at Ma = 0.1

(b) Lift at cell size 0.005

(c) Drag at Ma = 0.1

(d) Drag at cell size 0.005

Figure 5.25 Lift and drag coefficient simulated with the LBM over the time span of 8 s at several spatial resolutions and Mach numbers. Adapted from [158].

the other hand the LBM also exhibits many of the same characteristics of weakly compressible methods also seen with the FDM. This is the case with the high frequency oscillations induced by the non-smooth start that vanish after 2.5 s, as well as the oscillation at a much lower frequency. These also come from the compressibility-induced delay in reaching an incompressible state. Unlike the FDM however there is hardly any indication of vortex shedding at the peak of the inflow velocity. Only at the increased Mach number of Ma = 0.2 shown in Figure 5.25b very small fluctuations arise, which could be the result of vortex shedding. This matches the findings of the fully developed unsteady flow, for which the LBM introduces too much damping for the

expected vortex shedding at Ma = 0.1. However, overall the impact of different Mach numbers is small compared to the FDM.

The elapsed time for the simulations shown in Figure 5.25 are summarized in Table 5.23. The optimized code parameters used in these measurements come again from the steady flow case in Section 5.3.3.4.

5.4.2.5 Results

As noted earlier the comparison is performed in a different way than before, as not both methods can be tuned to optimal performance for the same target error at once, in particular as the lift and drag errors are difficult to determine exactly. This is especially true as the error changes over time. Also the necessity of matching the grid to the obstacle and the globally uniform mesh size make the possible increments in spatial resolution quite large and make it impossible to find a target error for which both approaches show optimal performance. The process is therefore reversed and the compared instances are matched by runtime, which are then contrasted for the simulation results. The full set of timing results for all tested configurations is given in Table 5.23.

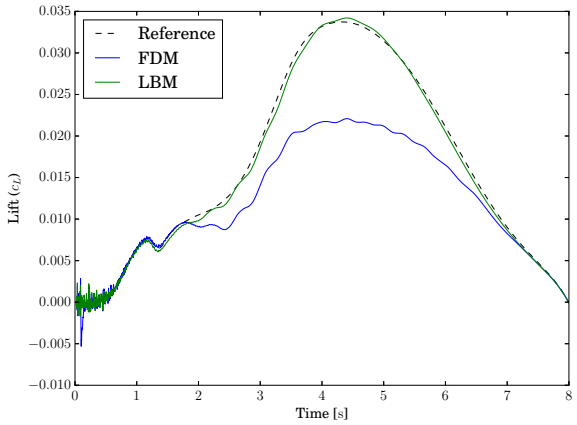
FDM								
Ma \ Δx	0.01	0.005	$0.00\bar{3}$	0.0025	0.002	$0.001\bar{6}$	$0.\overline{00142857}$	0.00125
0.2	337	6044	22 697	85 143	215 170	407 605	∘	∘
0.1	628	11 227	42 283	158 565	400 605	770 644	1 695 188	∘
0.05	1209	21 590	81 464	305 457	∘	∘	∘	∘

LBM								
Ma \ Δx	0.01	0.005	$0.00\bar{3}$	0.0025	0.002	$0.001\bar{6}$	$0.\overline{00142857}$	0.00125
0.2	-	581	2793	8769	23 450	34 911	81 025	∘
0.1	-	1161	5585	17 538	46 899	69 822	162 050	275 639
0.05	-	2322	11 171	35 077	93 797	∘	∘	∘

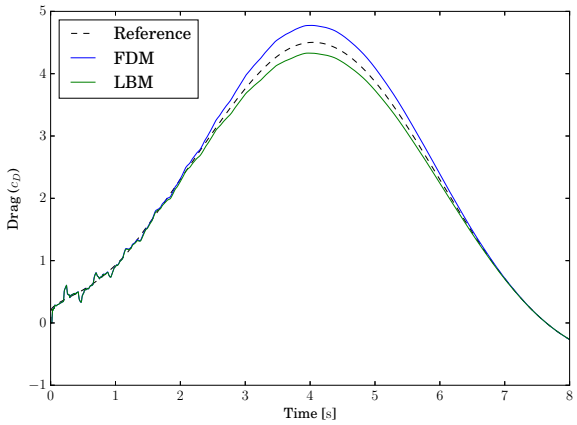
Table 5.23 Total runtime in seconds for various combinations of Ma number and grid spacing for the FDM and LBM. The marker (-) indicates no result and (∘) indicates the configuration was not tested.

The two sets of configurations are selected for their fairly comparable duration, which are within 10% difference of total runtime. The first combination is the FDM at Ma = 0.1 and $\Delta x = 0.00\bar{3}$ and the LBM at Ma = 0.1 and cell sizes of 0.002. The corresponding results are shown superimposed in Figures 5.26a-b. The second combination utilizes the FDM at Ma = 0.1 and $\Delta x = 0.0025$ and the LBM at Ma = 0.1 and cell size of $0.\overline{00142857}$ with their lift and drag coefficients as shown Figure 5.26c-d for the coarse and fine resolutions respectively.

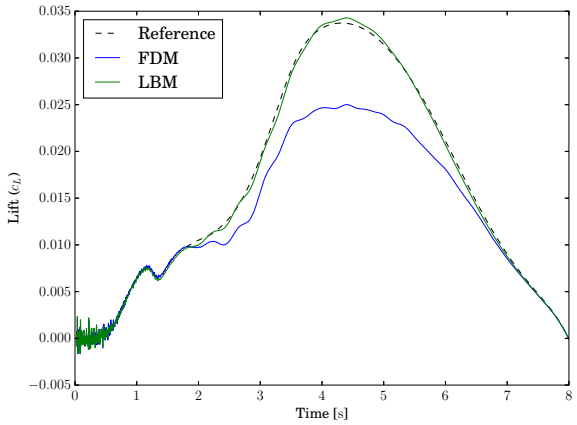
When analyzing the comparison of the shorter duration in 5.26a-b the LBM clearly shows a better match with the reference solution. Especially for the lift the difference is obvious, but also for the drag the LBM is at a small advantage. This becomes clear for the relative error of the maximum drag, which is 3.8% for the LBM and 6.0% for the FDM. However, the LBM lift coefficient solution overshoots past the reference solution even for this lower resolution. This situation worsens for the higher resolution version and means that the LBM reaches a fairly



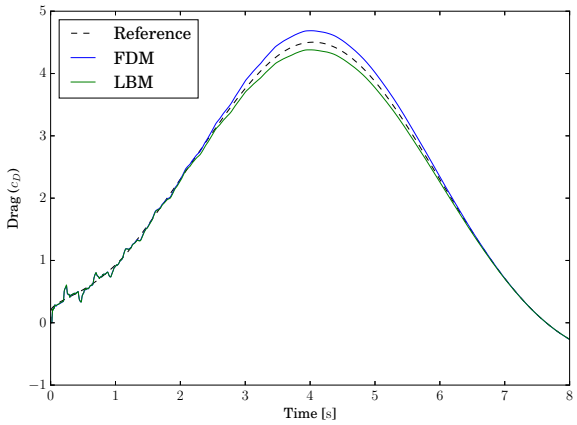
(a) Lift coefficient at simulation runtime of ~ 45000s



(b) Drag coefficient at simulation runtime of ~ 45000s



(c) Lift coefficient at simulation runtime of ~ 160000s



(d) Drag coefficient at simulation runtime of ~ 160000s

Figure 5.26 Lift and drag for FDM and LBM simulations at a comparable total run time. Adapted from [158].

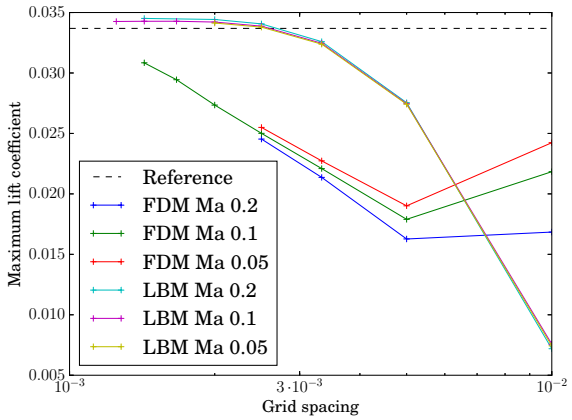
accurate result very quickly, but does not converge to the same solution as the direct solution to the Navier-Stokes equations with a higher order and high resolution implicit flow solver.

The comparison of the results for the second pair at the finer resolutions in Figure 5.26c-d is very similar to the first one. For the lift the LBM is again much closer to the reference solution, but the overshoot in terms of maximum lift is increased even further. Hence it could be considered a worse result then with the previous pair. However, with regard to the drag the LBM offers an absolute better solution and the maximum drag deviation is 2.7%. The corresponding relative error of the FDM is 4.1%.

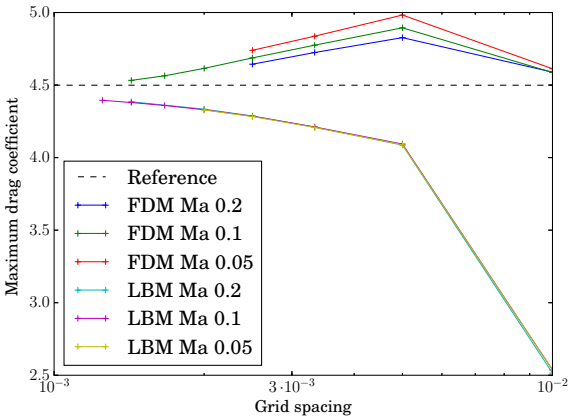
Apart from the error at the large scale the comparisons in Figure 5.26 also highlight other properties of the two approaches. Amongst others this is the noise at the simulation start and which can be traced back to the weakly compressible nature of both solvers. These high frequency oscillations are introduced by the non-smooth ramp-up mentioned earlier and are not present in the incompressible and implicitly solved reference solution. These acoustic modes also demonstrates the very high similarity of the behavior of both approaches despite the drastically different origins, as the oscillations are alike for the lift and almost identical for the drag.

Some form of quantification of these aforementioned errors would be required to give definite numbers to the performance of the compared methods. Indeed the error to the reference solution can simply be integrated over the simulation time, but this does not differentiate the error sources. For applications in engineering practice and therefore also this analysis, such a differentiation would be important, as noise surrounding the correct solution is much less critical than errors in the overall results. Perhaps these different error sources can be distinguished with different weighting and integration techniques, but this is outside the scope of this work especially as it has to be ensured that neither method is covertly discriminated against. An even bigger difficulty with the error quantification is the fact that the LBM solution best matches the reference solution for the lift coefficient at a medium resolution. Optimizing for the best result quality to runtime ratio must therefore also lead to these medium resolutions, which are of course not usable in practice when the solution is not known in advance. Typically it is expected that the results become monotonically better with increasing resolution, which renders such a mid way comparison not useful.

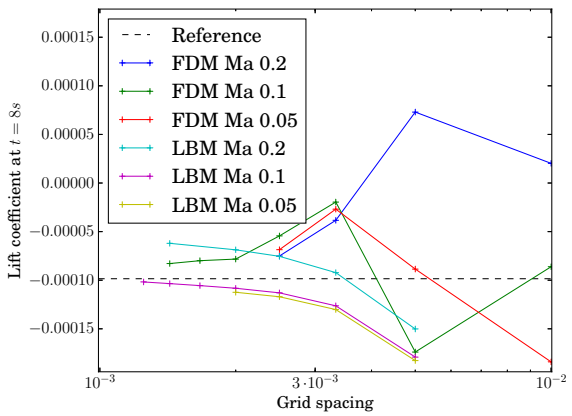
Instead several promising characteristic pointwise values are analyzed to find a quantifiable measure nonetheless. These are the maximum of the lift and drag, as well as their values at the end of the simulation, i.e. $t = 8s$, and their derivative at $t = 6s$. Using values in the second half of the simulation reduces the influence of the noise introduced by the ramp-up. However, even at that later state the convergence behavior of the different characteristic values is not particularly smooth. This can be seen in Figure 5.27. All lift-related measurements of the LBM converge to a slightly different solution than the reference result established with Adaflo. Even though the overall error to the reference is small, it causes the error in the characteristic value to vanish prematurely for intermediate spatial resolutions and then to increase again for finer lattices. This leads to the maximum lift and the slope of the lift at $t = 6s$ to be out of the question for establishing a direct performance ratio between FDM and LBM. Purely from a convergence perspective the lift coefficient at the simulation end point could be used for analysis. However, as this involves interpolating the spatial resolution to a desired error for which the noisy nature of the FDM solution is problematic. This is especially true for the fluctuating results seen in Figure 5.27c. The drag coefficient on the other hand shows a slightly different picture. As shown previously both methods overall approach the reference solution for the drag, albeit from different



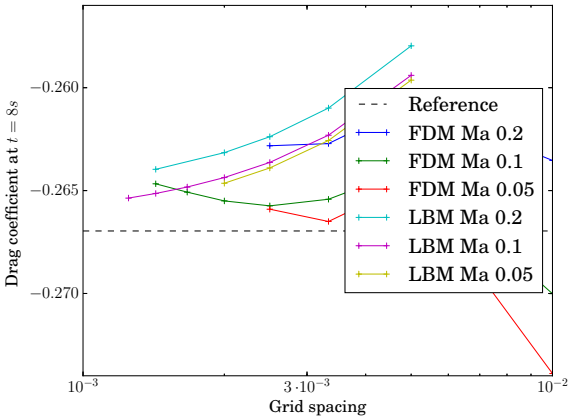
(a) Maximum lift coefficient



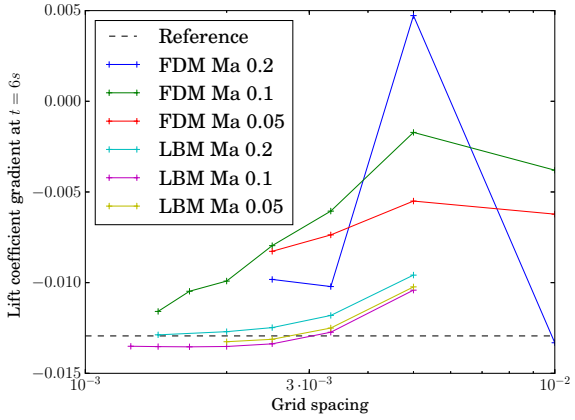
(b) Maximum drag coefficient



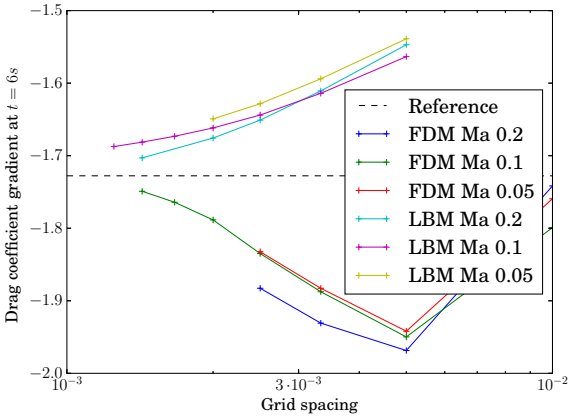
(c) Lift coefficient at $t = 8$ s



(d) Drag coefficient at $t = 8$ s



(e) Lift coefficient gradient at $t = 6$ s

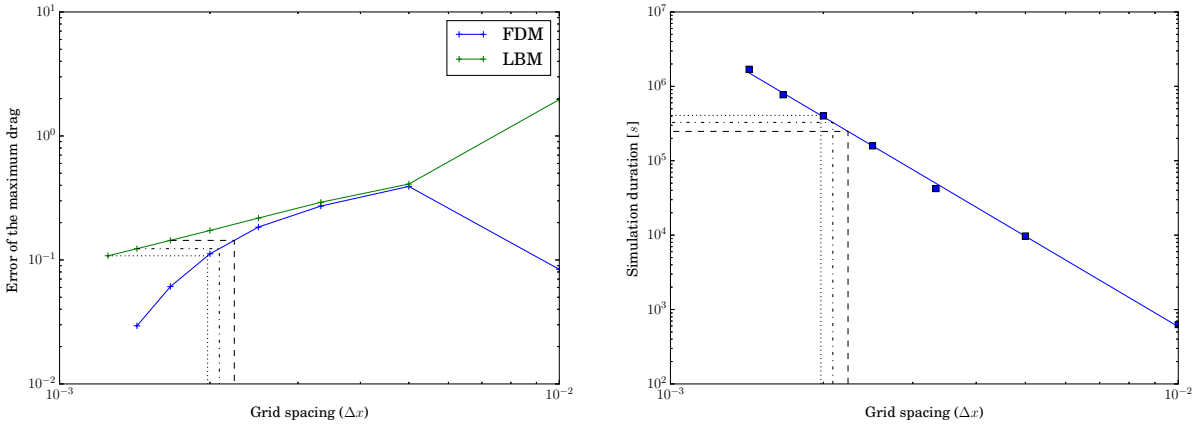


(f) Drag coefficient gradient at $t = 6$ s

Figure 5.27 Convergence of different characteristic values for the lift and drag coefficient as simulated with the FDM and LBM.

directions. A closer look reveals that this is not the case everywhere, as the drag coefficient at $t = 8s$ shown in Figure 5.27d does not converge for the FDM. The slope of the drag coefficient is generally more sensitive to the oscillations from which the simulations of weakly compressible type suffer than the direct values. This can be seen in the FDM results in Figure 5.27f, even though it is not as pronounced as for the slope of the lift coefficient. This leaves the maximum drag as the most suitable characteristic value to determine the performance ratio between both methods with respect to the incompressible reference solution. It is important to note that this only reflects a very specific single case and that the numbers given here cannot be applied to the performance of either method in general.

The optimal configurations for the maximum drag coefficient between both methods do not line up because of the large spatial resolution increments. Therefore the theoretically required resolution of one of the methods has to be interpolated. Since the LBM exhibits the larger error the matching FDM resolution is interpolated to give the identical error as demonstrated in Figure 5.28a. The resulting grid spacings are also given in Table 5.24. The corresponding



(a) Error of the maximum drag in relation to the FEM reference solution. The three LBM solutions on the finest mesh and the interpolated FDM equivalent are highlighted.

(b) Interpolated simulation duration for the FDM using a double logarithmic fit and the interpolated grid spacings.

Figure 5.28 The process of interpolated simulation timings for the FDM using the LBM's best maximum drag results. Adapted from [158].

simulation durations are obtained through a linear regression of the double logarithmic runtimes and spatial resolutions from simulations at Ma = 0.1. This fit matches the data points very well as can be seen in Figure 5.28b. The interpolated simulation durations are given in Table 5.24, which inherently include the required number of time steps for a given spatial resolution and hence the dependency on the Mach number. The results show a clear advantage for the LBM. This is especially true for the looser tolerance for which it is 3.6 times faster than the FDM. However, the data also suggests that there will be a break even point as the performance ratio reduces to 1.5 for the stricter tolerance. This behavior generally matches the findings from the previous test cases, but the comparison here is a very limited view of the overall performance. These performance ratios can by no means be transferred to the methods in general, as other characteristic values may result in completely different figures.

	LBM		FDM (interpolated)	
Error	Δx	Duration [s]	Δx	Duration [s]
0.1437	0.002	69 822	0.002 24	247 940
0.1233	$0.001\bar{6}$	162 050	0.002 09	327 492
0.1080	0.001 25	275 639	0.001 98	407 260

Table 5.24 The error in the maximum drag for the three finest resolutions of the LBM and the corresponding interpolation of the spatial resolution and simulation duration of the FDM.

The overall performance of the FDM and LBM for this test case is difficult to asses for a variety of reasons. The flow around a cylinder with a square cross section offers all properties sought after for a fair comparison as it allows the use of a matching grid, has sensible indirect engineering type measures with the lift and drag and leads to complex flow patterns. The drawback of course is its high sensitivity, which in turn requires very high resolution of the obstacle. The FDM was able to slowly approach the reference solution set forth by the higher order implicit solver to the NSE. The LBM leads to a slightly different behavior, which are amplified by the sensitive nature of the example. As a result convergence towards a different solution is seen. This makes quantitative comparisons very problematic because the method passes by the reference solution while refining the grid. The alternative of using reference results generated directly by the methods themselves, as was done for the lid driven cavity in Section 5.3.2.1 and the steady flow past a cylinder in Section 5.3.3, is not feasible due to the already resource-heavy simulations presented here. These simulations would have to be significant finer: At approximately 100 GiB memory consumption and runtimes of five days a further increase to the required level is not practical on shared memory machines. For this reason qualitative comparisons were carried out. These resulted in much better solutions of the LBM at same runtimes as the FDM. The strong influence of the resolution on the lift and drag coefficients acts an advantage for the LBM. As the required number of time steps is much lower than for the FDM, the higher resolutions can be handled faster and thus lead to a acceptable results with short runtimes. The drawback is of course that the LBM converges to a different result, which places a limit on the achievable accuracy. Approaching that limit is where the FDM catches up in terms of performance and eventually remains as the only option, barring alterations of either method.

To put the performance of the fully explicit, weakly compressible solvers, namely the FDM and LBM, into perspective, a short summary on the resource requirements is given and compared to an implicit and fully incompressible solver. The highest resolution simulation carried out with the implicit FEM solver using Q3/2 elements and three levels of local refinement lead to a minimal element size of 0.00079 and 447 million degrees of freedom (DOF). It used 1.2 TiB of main memory across 24 nodes and with a time step size of 0.01 s ran for a total of 4608 cpu hours. The highest resolution version simulated with the FDM at a grid spacing of $0.00\overline{142857}$ or 581 million DOF, an average time step size of 6.7 µs required 17 GiB of main memory and 3116 cpu hours. The highest resolution LBM simulation was executed at a cell size of 0.00125 which resulted in 872 million DOF in terms of velocity and pressure or 4142 million DOF in terms of density distribution functions. The average time step size was 32.1 µs with a main memory requirement of 90.8 GiB and took 2144 cpu hours. This shows that in the end it is a trade-off

between resources, time and solution quality and this is no different between the FDM and LBM as it is with other methods. The exact pros and cons are illustrated in the following section.

5.5 Performance conclusion

5.5.1 Performance summary

The five test cases presented in the previous sections have yielded interesting findings about the FDM and LBM, implemented in Fastflo and ILBDC, respectively. These findings are summarized for each test case in the following, with the focus on the special characteristics of either approach that are highlighted by each example.

For the steady duct flow an analytic solution is available and both methods have been tested against this solution of the Navier-Stokes equations. Under these conditions, and especially with the very simple boundary conditions that are required, both methods showed a similar runtime. However, there is a clear dependence on the strictness of the target error which is to be achieved. While the LBM is 2.08 times faster than the FDM for the coarsest tolerance, Fastflo is 4.35 times faster for the strictest tolerance. Hence, ILBDC is found to be very fast when solving to a coarse tolerance and resolution, but it increasingly struggles with the exact reproduction of the solution to the NSE. It requires much higher resolutions than Fastflo in order to be able to reproduce the analytical results with higher accuracy. The effect of a steep increase in simulation duration is further amplified by the interaction between the LBM and modern hardware. Modern hardware provides large caches, which can hide much of the high bandwidth requirements that arise from the 19 density distribution functions loaded and stored for every updated cell. This makes the LBM very fast for small domains. However, when the domain size grows past the caches' capabilities, the bandwidth limitation severely restricts the throughput and prohibits scaling with the large core counts found on many server platforms. The exact opposite applies to the FDM. It is better suited to reproduce the solution of the NSE exactly, as these equations are solved directly. This performance trend is further amplified by the code's scaling behavior. Due to the elaborate caching scheme, Fastflo only has to load four unknowns and update four RHS values per updated point. This leads to excellent scaling in particular for large domains and large core counts. However, it does take a large domain size for these techniques to show their strength, which leads to a relative performance penalty with the coarse solutions found in the duct flow example.

The non-leaky lid driven cavity is a much more complex test case than the duct flow. The durations and necessary resolutions are larger than before, which makes the differences in performance scaling between the two implementations more pronounced. This shift to larger domains is the main reason for the higher performance of the FDM in this example. The FDM is 1.25 to 2.17 times faster than the LBM for the coarse and strict tolerances respectively. This performance advantage is despite not evaluating the error against a direct reference solution of the NSE. An attempt at using such a reference, established by a higher order, inf-sup stable FEM, lead to a degraded spatial convergence of the LBM. The different application of the boundary conditions between the LBM and classical solvers of the NSE, for which an identical behavior is not guaranteed, necessitates other references for the error calculation. Both methods were compared to reference solutions generated by themselves on a much finer mesh, which is a much

more favorable testing condition for the LBM as it does not strictly have to reproduce a solution to the NSE. Yet still, the FDM outperformed the LBM in the tested range.

The test case of the LDC featured another measure for the result quality. It used the center plane of the channel to establish the number and position of vortices. Due to the irregular convergence behavior of the LBM in terms of vortex center positions, this indirect simulation quality measure was dismissed. The convergence behavior of the LBM showed similar characteristics to the comparison against the FEM reference solution for the velocity profiles. It reached an approximate solution rapidly, but then stagnated and fluctuated around the same result. It should be noted that the coarsest mesh exhibited an error approximately four times smaller than for the FDM and the error of the highest resolution simulation were of comparable magnitude between both methods. A further observation while testing different LDC configurations was that the indirect application of boundary conditions outside the domain with the many variations of Dirichlet conditions helped to handle singular flow configurations. At coarse resolutions when only a low accuracy result can be expected the LBM managed to produced fairly accurate solutions nonetheless.

The concept of the indirect or engineering measures was picked up again for the test case of a steady-state flow around a cylinder with a square cross section. The lift and drag coefficients of the obstacle are used in this regard, as well as the pressure difference across the cylinder. Without the direct comparison of velocity and pressure, the methods only have to achieve the most accurate results directly in terms of the final quantities. This eliminates the need for an intermediate step of a highly accurate velocity or pressure field, which may not always be given for a particular test case, but which can still result in very accurate postprocessed measures. The overall outcome of this comparison is inconclusive. Additionally, a direct comparison with each method at its optimal configuration is prevented by the large resolution increments. These are due to the geometry and the restriction to matching grids. However, a conclusion, which can be drawn is that both methods show a very similar convergence behavior and, actual result quality aside, the LBM is faster at the same spatial resolution due to the larger allowable time step size. Establishing which approach is actually faster in terms of solution quality per runtime is more involved. Firstly, the previously used comparison technique for which both approaches are tuned optimally to the same error is not possible and approximations have to be made. Secondly, the performance question has to be investigated for all three quantities of interest separately, as the results are very different. In this context, the most accurate result for the lift coefficient obtained through the LBM was compared to an approximation of the corresponding FDM result. The conclusion was that Fastflo can utilize a lower resolution and thus be at least 12.7 times faster than ILBDC. However, for the pressure difference the opposite was the case as the employed FDM implementation struggled with the pressure leading up to the obstacle, meaning that the LBM can be at least 586 times faster. For the drag coefficient, as the middle ground between the lift coefficient and the pressure difference, the achievable tolerances per resolution coincided for both methods, ensuring they were near their respective optima. It was concluded that for the selected tolerance of 10^{-2} the LBM was 4.8 times faster. It is not surprising that the performance ratio for the drag coefficient is in between the other two results. The drag is composed of the pressure difference component, which the LBM recovers more quickly and the traction on and below the obstacle, which the FDM evaluated more accurately. Overall, neither method could demonstrate comprehensive performance advantages for the steady-state flow around a cylinder with a square cross section.

The unsteady flow past a square cylinder is a very sensitive example. This was of particular importance for the unsteady flow that utilized the vortex shedding behind the obstacle to assess the performance. The combination of this sensitivity with the long simulation durations necessary to reach the fully developed state made a performance comparison impractical. This holds especially true as the coarser resolutions displayed erratic behavior with a wide spread of different results for the lift and drag oscillations and much higher resolutions would have had to be used. Another point that was moved into focus through this example is the weakly compressible nature of both flows. In the context of time-variant flow this becomes important and can be difficult to handle, but inherently comes with the fully explicit nature. Due to the lack of proper performance assessment opportunities, this example was used to evaluate the influence of the simulation parameters and to find a configuration suitable for time-dependent problems. For the FDM this was a simple process, as it showed the desired behavior at $\mathrm{Ma} = 0.1$ and with sufficient spatial resolution. The investigation also uncovered difficulties with increased damping in LBM. It required additional excitation with $\mathrm{Ma} = 0.2$ to induce vortex shedding. However, this is not critical for the next test case, so both approaches can make use of an identical Mach number configuration.

The flow past a cylinder with a square cross section was slightly more forgiving for the transient test case, where the inflow velocity is slowly ramped up and down. The fixed simulation time of 8 s naturally limited the simulation runtime, which allowed for much higher resolutions to be analyzed in a reasonable time frame. This in turn lead to a much more complete and distinct spatial convergence analysis. A reference solution was again obtained through a very high resolution, higher order, implicit finite element simulation. This gave a reference for the expected behavior from which both the artificial compressibility FDM and the LBM deviated for multiple reasons. Quantifying these errors in an unbiased way is difficult and should distinguish the individual error sources as they have different importance in practice, e.g. the acoustic noise from the non-smooth start may not be important when establishing the maximum force on the obstacle, whereas the deviations in the maximum lift and drag are essential. However, such differentiation can introduce hidden sources of discrimination against either method. To still compare both approaches without a reliable quantification of the result accuracy, the comparison process was reversed and simulations with similar runtimes were compared visually. This was easily done as the LBM offered a significantly better solution for the lift coefficient in both compared combinations. The difference was less apparent for the drag coefficient, as both methods lead to very similar results, even including artifacts from the compressible nature of the fluid. However, for the maximum drag coefficient the LBM produced slightly better results at 2.7% relative error, rather then 4.1%. While this could have lead to the conclusion that the LBM shows overall superior performance, further investigations uncovered a major drawback of the LBM. It overshot past the reference solution for the lift coefficient and, as with previous test cases, it does not converge to the NSE solution. This makes it clear that the accuracy of the LBM in terms of providing a solution to the NSE is limited. Past this limit the FDM is the better option. The point where this occurs exactly is a matter of the QOI and the way the error is evaluated. To be able to quantify the performance difference at least at some level, a series of characteristic values were analyzed. These measures were designed to be the least problematic in conjunction with the weakly compressible nature of the simulations and of these the maximum drag coefficient was selected as the most suitable candidate. By using the best result the LBM offered for this quantity and interpolating the hypothetical resolution and simulation duration of

the FDM, a performance ratio has been determined. The LBM was faster by at most a factor of 4.7 to 2.3 for a larger and a smaller error. As this result was obtained for the optimal LBM setup, but only the interpolated FDM results, the difference could be smaller in reality.

Of course the big question of which approach is faster overall remains unanswered. There is, and can be, no absolute answer to this question, unless every possible combination has been tested and compared. Even then, there would be areas for which either approach is more suited. The distinction of areas in which either one or the other method excels is discussed in the following section, along with not strictly performance related pros and cons to either technique.

5.5.2 Suitable use cases

As the investigations presented above have shown, it is very difficult to directly quantify the performance of the FDM and LBM or in fact any other method. For very simplistic examples it is indeed possible and even in a fair manner, however these findings cannot be directly transferred to problems closer to everyday engineering practice. The insights from the more complex examples uncovered some general trends, which may not allow the performance to be quantified specifically, but enable a categorization of both methods into problem settings for which they are best suited.

The most important categorization, which has shone through in all the test cases above, is the distinction in loose or strict tolerances. The LBM is remarkably fast to a solution that roughly matches either a direct solution to the NSE or an LBM solution on a much finer mesh. With stricter tolerances however ILBDC requires increasingly higher spatial resolutions, which in turn induces additional penalties through the severe bandwidth limitation and which allows Fastflo to eventually catch up. The FDM's higher evaluation cost at comparable spatial resolutions is mostly caused by the high number of required time steps. It does however pay off by being able to reach much smaller errors. What exactly constitutes a large or small error and when the break even point is reached, is a question of the problem to be simulated, the measured quantities and the employed hardware.

The distinction into suitability for strict or loose tolerances can be continued further since the previous tests have shown that an indirect measure also aids the LBM. Small deviations from the exact solution to the NSE can be balanced out, reducing their impact on the quality assessment. This caused the performance question to shift somewhat in the favor of the LBM for the flow around a cylinder example, where the QOI of lift and drag were used. Meanwhile the FDM showed superior performance for direct assessments of the error in velocity and pressure profiles, preferably against an analytical solution to the NSE.

The issues of the LBM with stricter tolerances could potentially be improved by using different formulations or more importantly using different and more accurate boundary conditions. There are many variations of boundary conditions for the LBM to choose from, even for straight forward no-slip Dirichlet conditions. This is caused by the prescription of boundary conditions at the velocity and pressure level to be applied in a weak sense, which are therefore open for many different formulations. It gives rise to one of the soft factors influencing the decision in favor or against the LBM. The wide selection of BC choices, even for boundaries that are straight forward for traditional NSE based solvers, adds complication to the setup process. It requires balancing accuracy improvements against higher computational cost, which can be difficult to assess in advance and are highly case dependent, as was also shown in this work. Nonetheless this complexity in the boundary conditions goes hand in hand with a simpler use within the domain.

Unlike the FDM no stabilization techniques are required and a robust solution at an ideal time step size is ensured within the stability range. This creates a trade off between complexity within the domain or on the boundary conditions. Aspects in favor of a simpler domain evaluation is that the domain discretization size typically grows with a power of three and the boundaries surfaces only with a power of two. From a practical point of view however, the true effort lies in the number of boundary conditions that have to be set up and which is up against a single inner domain. On top of that the configuration of BCs tends to be much more complex, especially in corner cases where different types of conditions meet and their interaction has to be coordinated, compared to the usually smooth flow throughout the domain with few tolerant stabilization parameter. It has to be noted that these considerations only hold for the plain and high performance techniques used here. For either method more complex techniques promising higher convergence orders and smaller error can be used, at the expense of the high update rates achieved with the presented approaches. Some of the possible extensions could be parameter free stabilization or immersed boundary conditions for the FDM or configurable multi-relaxation time or non-linear collision operators for the LBM.

A further difficulty that arises for the LBM in this context is the ease of interpretation. The flow behavior cannot be directly interpreted from the density distribution functions and the process of recovering the velocity and pressure can hide problems at the density level. This is particularly problematic for the already complex application of boundary conditions as their interaction in the corners is difficult to interpret and issues are hard to resolve.

Overall neither approach is without problems, which is to be expected as the focus on high performance naturally leads to some drawbacks in other areas. Some advantages of the artificial compressibility FDM over the LBM is the faster solution of the NSE at stricter tolerances and smaller errors. Also complex boundary condition combinations are easier to set up due to the inherently simpler conditions and the direct use and interpretation of velocity and pressure. The LBM on the other hand is very fast for coarse solutions. It is often applied to porous flows or through small and complex geometries where the results do not have to be exact in a point wise sense, but rather for phenomena over larger parts of the domain. The involved geometries are typically too complex for body fitted meshes, requiring immersed techniques that typically reduce the accuracy anyways. The flow can also often not be fully resolved and thus the relevance of the small deviations is further reduced. Additionally QOI gathered over large portions of the domain can balance out these deviations further. However, even if only few of these conditions are fulfilled or very coarse results are sufficient the LBM can be a viable and fast option.

6 Summary and outlook

6.1 Summary

The goal of this work was the comparison of fast CFD solvers and quantifying their performance in engineering terms. These performance terms were a predetermined rough solution quality in the shortest time or the best solution in a given time span. This differs from the performance figures typically found in literature, which either show high convergence orders or high update rates. However, only the product of both makes up a fast performance in a practical sense. Additionally, the absolute error, the choice of the test case, the optimization level of the implementation and more come into play and were subject of analysis here.

Due to the countless options when establishing fast CFD methods the selection process for the methods to be compared is just as important as the comparison itself. This is largely caused by the excessive number of possibilities, which cannot be explored exhaustively. The selection process ultimately determines the domain in which the tested approaches have demonstrated their performance and within which boundaries it is valid to draw conclusions. Many of the necessary restrictions can give an advantage to a particular method, which further increases the importance of a clear and openly presented selection process.

In order to structure this overwhelming width of possibilities this work has divided the procedure into three stages: the method selection, the code tuning, and the application to specific test cases. Naturally, these stages are not fully independent, as for example the test case has a large influence on which method will perform well. Instead of presenting the ensuing iterations for tuning to a test case, the question was posed the other way around. It therefore became a matter of which test case suits the methods under consideration and how can their performance differences be accentuated.

The selection of the methods for comparison had to consider the wide and diverse field of CFD simulation techniques. Here an artificial compressibility finite difference method and a lattice-Boltzmann method have been used, as these are not only very fast solvers in their usual field of application, but also offer other interesting comparison opportunities. The close similarity in both methods behavior and yet radically different approaches allowed for more insights than purely comparing absolute performance figures. With the LBM having a reputation of particularly high performance, the comparison was also carried out under the aspect of whether this stems from the lattice-Boltzmann approach itself or if other factors, that may be transferred to the FDM, are decisive. The steps that were taken to perform a fair comparison and the findings whether there were performance differences between the LBM and FDM are summarized in the following. Additionally, the question of the origin of the reputed high performance of the LBM has been investigated.

The precise configuration of the methods was first carried out for the LBM. Due to the longer history of the FDM most behavior of the LBM can be matched with corresponding FD techniques.

Based on theoretical considerations, experience, and known problems with particular lattice-Boltzmann variants a two-relaxation time approach has been chosen. It is simple and can be optimized well, which ensures high update rates. At the same time it avoids some of the major drawbacks of single-relaxation time models. The boundary conditions have been selected with similar considerations. They were implemented as mid-way bounce back schemes, which are very simple, fast and integrate well with the overall indirect addressing implementation at marginal cost. The combination with the TRT scheme also eliminated some sources of inaccuracy with the boundary condition. The finite difference method was specifically chosen to match the LBM. On the one hand this promised high performance as many of the performance characteristics of the fast LBM are matched and on the other hand it made it possible to compare both methods with regard to the underlying approaches, by eliminating as many undesired influences as possible. This has led to an artificial compressibility approach with a low storage, two stage explicit Runge-Kutta time integration and a simple second order stencil with artificial viscosity stabilization. It is of similar convergence order and behavior as the TRT LBM and also lends itself well to tuning.

In the tuning stage the implementations of these methods have been optimized. The tuning process was very important, as the speedup of 2.15 for the LBM and 4.03 for the FDM over an unoptimized but performance oriented parallelized version showed. To achieve these results a wide variety of optimization techniques had to be applied and tested for their effectiveness. Some of these techniques showed performance gains only in combination. To verify their effect performance models were required. These used were paired up with tools to analyze the impact of the various techniques on many metrics, such as memory bandwidth usage and instruction throughput. Three performance models with varying levels of detail and accuracy were used with some success. However, even the execution-cache-memory model, as the most detailed one, struggled to give precise estimates for the highly tuned code on complex modern hardware. The investigation into the cause of the inaccuracies found that not a single reason was the cause, but rather that every increase in tuning and more complex hardware added some error. This eventually resulted in a factor of two between estimate and measurement. These large errors occurred despite intricate knowledge of the compiled code and hardware specifications. Improvements can be achieved by inserting correction factors derived from measurements, but ultimately this makes the predictive benefit from such a model very small as it simply reproduces the measurements. Nonetheless the performance models are very useful in the iterative process of tuning, comparing estimates to measurements, and eliminating discrepancies by either refining the model or improving the optimization.

To overcome the performance gain limitations that persisted even with this iterative tuning process, it was followed up by an extensive brute force search of the optimal code parameters. The results were the two to four fold speedups mentioned earlier. In terms of absolute performance the LBM loop kernel was found to be capable of 218 million updates per second, but was held back by its high memory bandwidth requirements leading to an early bandwidth saturation. The FDM loop kernel in its fully optimized configuration had a balance of bandwidth and arithmetic requirements very suitable for the reference platform. It showed excellent scalability reaching a maximum of 768 million updates per second. Of course these numbers cannot be transferred directly to the practical performance as they were measured in a benchmarking framework and with domain sizes utilizing the full memory resources. It also disregarded the actual performance in terms of runtime to a specific solution quality.

With highly optimized codes created, the third stage of the comparison process was carried out. The test case selection and runtime comparisons were arguably the most important step when establishing the CFD performance characteristics. As mentioned previously, it is infeasible to test every possible method that might perform best for a predetermined test case. Instead the test cases were selected to suit the available methods. Naturally, this means that highest performance cannot be shown universally, but it is suitable to the comparison between the LBM and FDM. The tests were designed to highlight the differences between the two approaches and to determine their advantages and their performance break even point. Additionally the tests were designed to increase in complexity to cover simple analytical solutions up to setups that resemble practical problems.

As the first test case a duct flow was chosen for its simplicity in terms of boundary conditions, as well as the analytical solution that minimizes any bias towards either approach. By measuring the velocity and pressure profile in the downstream section of a fully developed flow, the impact of differences in the interpretation and exact application position of in- and outflow conditions was reduced. The compactness of the example also allowed for extensive optimization of every configurable simulation and code parameter for every considered target tolerance. The performance tuning was carried out until improvements were on the same magnitude as the noise from the time measurement. The obtained performance can therefore be considered optimal for the given combination of methods and test case. The results showed a trend that was also observed with all other test cases. The LBM performance was better for coarser tolerances, here by a factor of 2.1, and the FDM was up to 4.3 times faster for smaller target errors.

A similar behavior was discovered for the second test case of a non-leaky lid driven cavity. It was chosen due to it prevalence in literature and the much more challenging flow pattern, compared to the duct flow. Despite the simple setup some challenges in finding a fair configuration had to be resolved. For instance a non-leaky variant with a special lid velocity profile had to be used to ensure an identical interpretation of the BCs for both approaches. In addition to that, no analytical solution is known and the reference solution obtained with an implicit, higher order FEM lead to a degradation of the convergence behavior, mainly of the LBM. In order not to penalize the LBM on the grounds of an uncertain reference solution, the measurements were carried out with references generated by both methods themselves at a much higher resolution. As with the first test case all simulations and code parameters were tuned extensively to optimal performance and the quality of the solution was measured as the relative error over the velocity field. In this case the FDM was between 1.25 and 2.17 times faster than the LBM over the whole range of tested target tolerances. As before, the FDM showed an increasing performance for stricter errors and higher resolutions. Apart from this rigorous error quantification through the velocity field, indirect measures were tested as well. These reflected a scalar by which an engineer might rate the design. However, the vortex center position in the mid plane of the cavity turned out to be ill suited for a performance quantification. This was because the vortex position error for the LBM started out much smaller, but quickly leveled of at a plateau around which it fluctuated. This inhibited the generation of fair performance ratios, but the qualitative results conformed to the previous findings.

The third test case was the final steady-state configuration, which also acted as a precursor to the time-variant tests. It was an enclosed laminar flow around a cylinder with a square cross section. This offered the opportunity to use some different quantities for the simulation quality quantification. The used lift and drag coefficient as well as the pressure difference across the

cylinder were a further step towards engineer practice. In line with practice-oriented procedures, the simulation tuning process was simplified. By using a single set of parameters independent of the target tolerance the efforts for the tuning step were reduced to manageable limits. As opposed to the simulation parameter, the code parameter were optimized and tabulated for the few possible domain sizes. The same set of simulation and code parameters was also reused for the time-variant versions of this test case. The results mostly agree with the results published in [129] with the drag coefficient for the LBM slightly above the given range. The flow around a square cylinder was very sensitive and the published values exhibit large variations. Interestingly the reference values that were established with a LBM exhibited a similar effect as the LBM in this work. Due to the large differences between potential reference values for quantifying the result quality, the comparison was again performed against high resolution simulations of the FDM and LBM. However, there were some difficulties that hindered a direct comparison of the approaches, as done in the previous examples. Most importantly the increments between usable resolutions were very large, which made it a matter of coincidence whether the methods combinations of ideal configuration and target error matched up. Additionally the three measures considered here gave very different tendencies as to which method performed better, making it problematic to judge on the overall performance. A comparison by the spatial resolution and disregarding the quality of the results, showed a higher performance of the LBM. A much more meaningful result was obtained by interpolating the more capable method to the smallest achieved error by the more limited one. This gave at least an upper bound on the performance differences. The FDM was found to be at least 12.7 times faster with regard to simulating the lift coefficient, while the LBM was 4.8 and at least 586 times faster for the drag coefficient and pressure difference respectively.

The time-dependent simulations were a somewhat double-edge sword for the weakly compressible methods used here. On the one hand the explicit time integration allows for a very high throughput of time steps, while on the other hand neither of the two approaches is time accurate. However, in a suitable configuration they can still yield good results, in particular with the coarser tolerances considered in this work. Hence, the first time dependent test case was selected to be the unsteady flow past a cylinder with a square cross section. The unsteady behavior was a vortex street behind the obstacle. The comparisons were carried out for the fully developed state using the same quantities of interest as for the laminar case and additionally the Strouhal number. However, no precise reference solution were available and the values provided by literature are sparse, but generally in agreement with the simulation results. Establishing a reference solution with simulations at a much higher resolution was infeasible due to the long runtimes required to reach the fully developed state. With these very high number of time steps practicality limited the spatial resolution to very coarse discretizations. The results obtained from these coarse simulations did not exhibit proper convergence, as for example the oscillation frequency in single QOIs changed drastically or were phase shifted. No conclusion regarding the performance were possible, but some helpful insights on the suitable Mach number selection for the transient test case could be made anyways.

As the last test case the flow past a cylinder with a square cross section was used in a transient configuration. The time dependent inflow created additional demands on the compared methods. In contrast to the unsteady case the flow did not have the time to fully develop. Instead the simulation was run for a fixed simulation time period. This added errors in the form of shockwaves induced by the non-smooth start ontop of the general compressibility effects. For the

comparison the lift and drag coefficient development over time was used and again the optimized simulation and code parameter from the laminar case have been used. However, much higher resolutions than for the unsteady example were possible, as the runtime was limited by the 8 s simulation time and which is considerably shorter than it takes for the vortex street to fully develop. The same FEM implementation as for the lid driven cavity was used to provide a truly incompressible reference solution. Most of the comparison had to be carried out qualitatively against this reference, as the difficulties with the large increments in domain resolution persisted. Additionally the noisy start with acoustic modes hindered accurate comparisons over the entire time span. Overall the results of the LBM and FDM were very similar and their behavior, especially with regards to the compressibility artifacts almost identical. In order to give some quantification of the performance difference the process from the initial test cases was reversed. Instead of comparing the timings at a set precision, simulations with comparable duration were investigated with regard to solution quality. Even without precise measurements this analysis determined a clear performance advantage of the LBM. The attempt to find other metrics suitable for performance measurements showed that convergence for most characteristic values were not smooth and the LBM even surpassed the reference for the maximum lift. The only candidate that allowed the performance to be estimated by smoothly interpolating the required resolution and simulation runtime was the maximum drag. Interpolating the total time to solution for the FDM to the LBM results yielded a factor of 4.7 by which the LBM outperformed the FDM. However, when the same process was carried out for the highest resolution simulation the factor reduced to 2.3. As with all previous examples this test case showed that the strength of the LBM lies at larger errors and coarser meshes. And while the FDM made up some ground for the higher resolutions it was not able to outperform the LBM under these specific circumstances. The use of even finer meshes to eventually see the FDM outperform the LBM in this example is impractical due to hardware limitations.

The overall question of whether the LBM or FDM is faster could unsurprisingly not be determined universally. In general, the data showed very similar behavior. This similarity transferred to the performance, which was also found to be comparable. It has to be concluded that it is not the underlying approach that causes the high performance of the LBM, but rather a by choice restricted feature set and the excellent tunabillity that comes with it. Application of the same principal to the FDM yielded comparable performance. However, some general trends in the performance figures could be observed. For all test cases the LBM performed better at coarse tolerances or for indirect measures, while the FDM preferred stricter tolerances and was able to reproduce the direct solution to the incompressible NSE more closely. Ultimately the question of which method performed better was always a matter of the exact configuration of the test case and the target tolerance.

Further findings were secondary factors that came up during testing and which can also influence the decision for or against a method. The FDM requires stabilization of the stencil operator, which can be difficult to configure suitably, while the LBM is very straight forward in this regard. The LBMs drawback come from the boundary conditions, for which many alternatives are available, each with its own advantages and drawbacks. The difficulty with the boundary condition application is further intensified by the use of the many density distribution functions that cannot be interpreted as easily as a direct use of velocity and pressure.

6.2 Outlook

A large portion of this work was the reduction of the number of parameters and configurations to magnitudes that can reasonably be handled. Many choices and restrictions for a multitude of reasons had to be applied to achieve this reduction. Most of them lend themselves well to further investigation. Of course changes to some of the fundamental choices would lead to entirely new works rather than extensions to this one. Furthermore, not all of them lead to relevant results that are worth investigating. Some decisions may be apparent and do not result in new insights or they may be conflicting. It was one of the objectives of this work to omit exactly these parameters in the first place. In the following some ideas for promising continuations of this research are given. As there are a large number of possible extensions, it is divided in the familiar three categories of choosing methods, optimizations and test cases.

Certainly the biggest impact on the scope and completeness of the investigation of fast CFD has the selection of methods. Even fundamentally different approaches could be compared to one another, however, the findings must remain relevant. Showing apparent differences between methods is of little use. The advantages and disadvantages of methods that are used for similar purposes offer much more interesting insights, in particular their break even points. In this spirit an extension of this work to an even clearer separation of the LBM from the FDM offers many opportunities. A topic of ongoing discussion is the influence of temporal and spatial convergence orders on the performance. A comparison with thoroughly optimized code and from an engineering perspective could offer interesting insights. This is especially the case in conjunction with the LBM as a truly independent reference to compare to. Naturally the LBM also offers many possibilities of extension to more complex multi-relaxation time or even non-linear schemes, improving stability and accuracy and reducing the throughput. This leads back to modifications of the FDM loop kernel, as especially the stabilization offers countless possibilities to test. There may potentially even be faster ones than the artificial viscosity used here. Also the question if a parameter free stabilization is an advantage or disadvantage is interesting. The additional effort of tuning the stabilization parameters must of course pay off in terms of performance. Perhaps also the LBM performance could benefit from additional tuning parameters, as for example required in the application of some boundary condition methods.

The optimization does not offer that many possible points of extension. This is largely because the possible tuning techniques are essentially prescribed by the used methods and hardware platform. Of course there are still minor improvements possible, however, the gains tend to become marginal and require unreasonable levels of effort. This can easily be verified through the performance models which can reliably provide the performance ceiling under ideal conditions. Additionally these highly optimized techniques tend to obstruct any portability and flexibility for method modifications. A completely different strategy would be the use of automated tuning for both methods. This would take the manual search for ideal constellations out of the equation. However, even then the influence of some of the very basic performance programming, such as memory layout, would still remain and could skew the results undetected. As the tuning is very closely bound to the hardware an extension of this work to further architectures is also relevant. The topic was already addressed in this work, but with the recent broadening of available hardware architectures it is likely to become even more relevant in the future. This is reflected in the current supercomputer landscape which shows a wide mix of architectures and the top ten supercomputer worldwide already cover the spectrum of general purpose CPUs with and without

accelerators, such as GPGPU or MIC designs, with streaming architectures or even fully custom architectures [1].

The test cases, as the third category, offer many more opportunities to expand on than the optimization. Theoretically there are infinitely many configurations. Here however, some very specific suggestions on the most promising additions are made. The most obvious is the continuation of the sequence of test cases with increasing complexity used in this work. These problem settings have represented some of the engineering tasks found in practice, but it falls short of covering the bulk. A very important prerequisite to more elaborate examples is the incorporation of more complex geometries. This requires techniques like immersed boundary conditions, which opens up a vast set of additional performance investigations. Naturally, this ties back into the expansion of the used methods described earlier. It is likely the point with the single biggest impact on relevance to engineering. With a departure from the practice-orientation in some regard, the test cases could also be expanded to analyze setups relevant for so called grand challenges, which are mostly tackled by research institutes or universities. All examples used in this work have been designed to suit both methods well as not to create a bias towards either approach. In the context of distinguishing the difference between the LBM and a similar FDM, test cases that are challenging for both approaches in equal amounts could offer interesting insights. It would emphasize the flexibility and good-naturedness of both approaches. Of course the test case selection can also be expanded with regard to flow type and regime. For example the limitations of the methods can be tested by varying the flow regime to high Reynolds numbers and by extending the approaches with turbulence models. Alternatively low Reynolds numbers and flow through porous structures could be investigated. This is especially interesting as the LBM is particularly popular for porous flows.

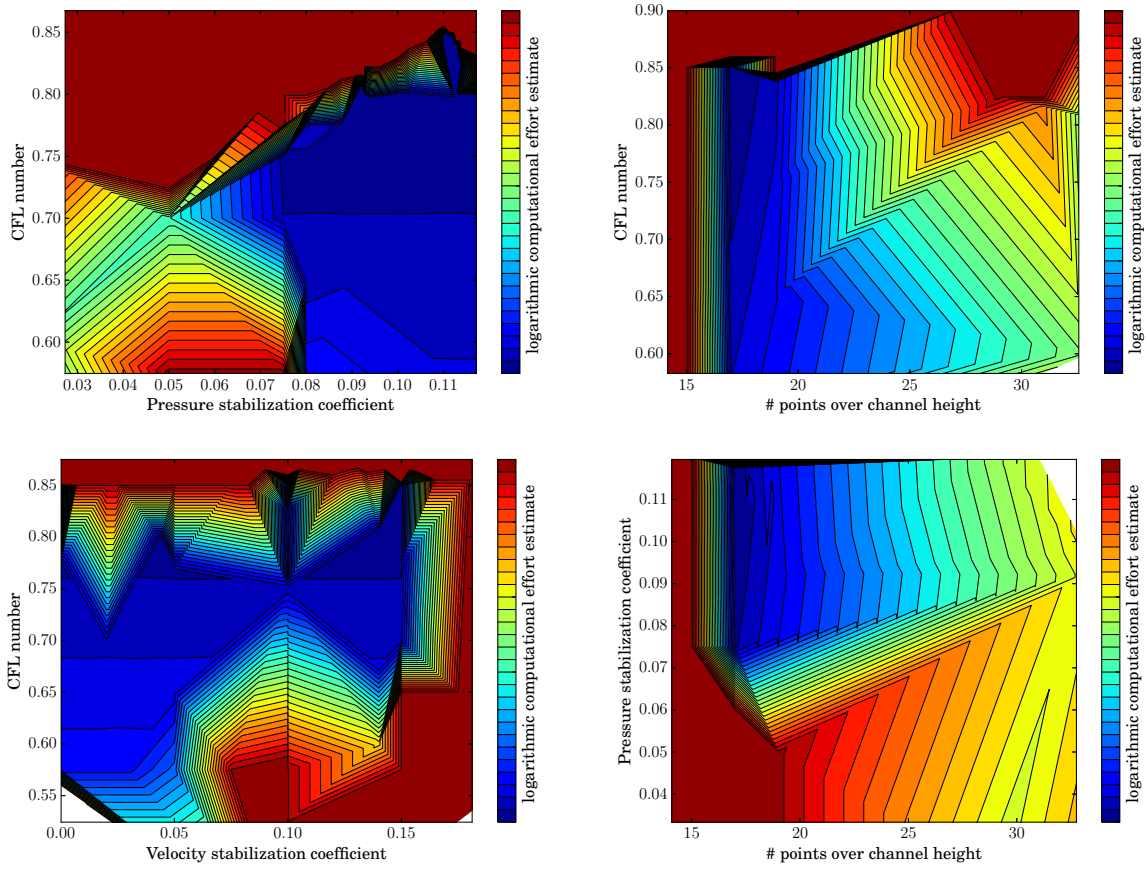
Finally, no matter how many additional methods, hardware architectures and test cases are examined, the topic of finding the fastest incompressible CFD solver is simply too wide to cover substantial portions. Even finding the fastest method for a particular problem cannot be achieved with certainty, as testing every possible technique and parameter configuration is orders of magnitude more effort than feasible. This problem is greatly exacerbated when the test cases are varied as well. However, in practice very high performance is usually sufficient, which means that when enough options have been tried the performance will likely be good enough for almost all purposes.

In this context the question of whether the LBM is really that fast or at least faster than a similar FDM can also not to be answered definitely. First of all their performance is too similar and secondly it always depends directly on the circumstances, which can favor either approach. This means that the strengths can be differentiated further and the preferred areas of application can be outlined more closely, but the question of which methods is faster will always depend on the exact set-up.

A Full simulation parameter optimization

A.1 Duct flow

The full set of simulation parameters for the duct flow at a target tolerance of $\sqrt{10} \cdot 10^{-4}$ given in most pairwise combinations. An excerpt is used in Figure 5.4a.



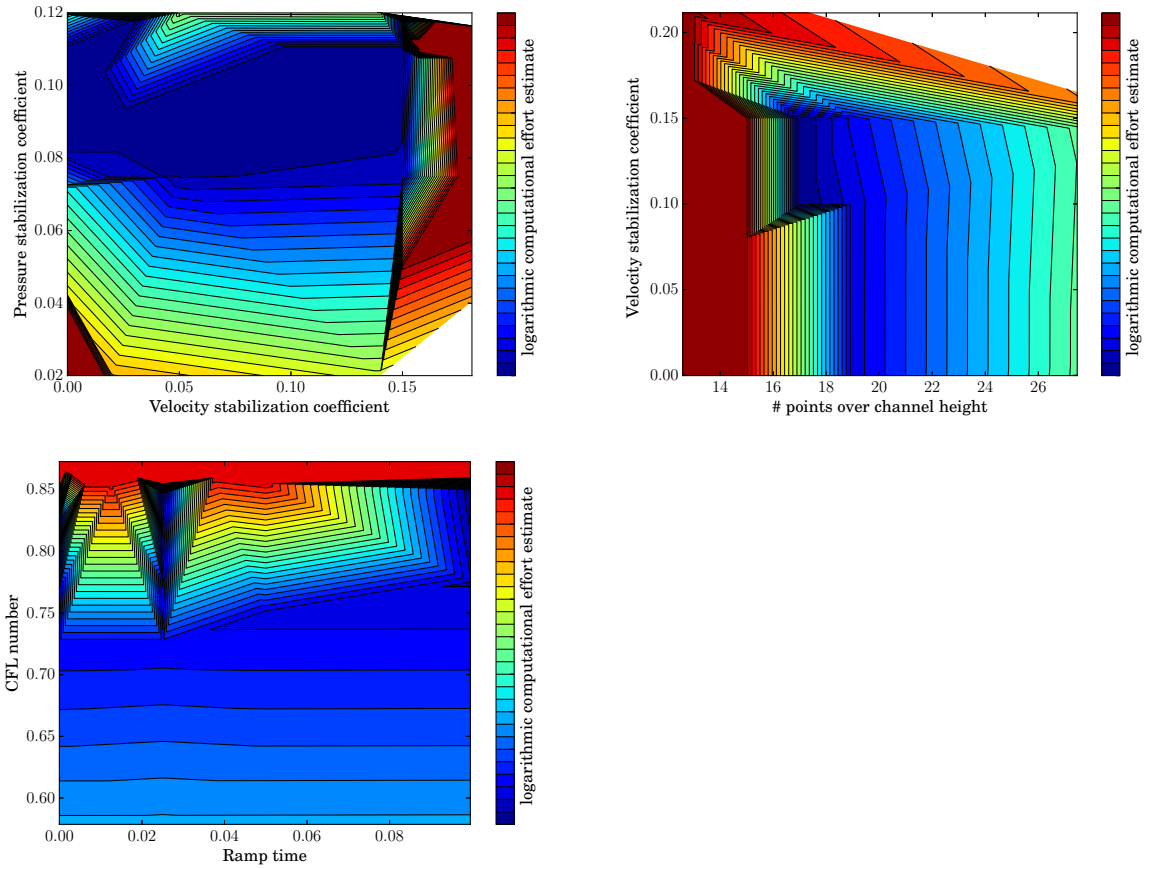
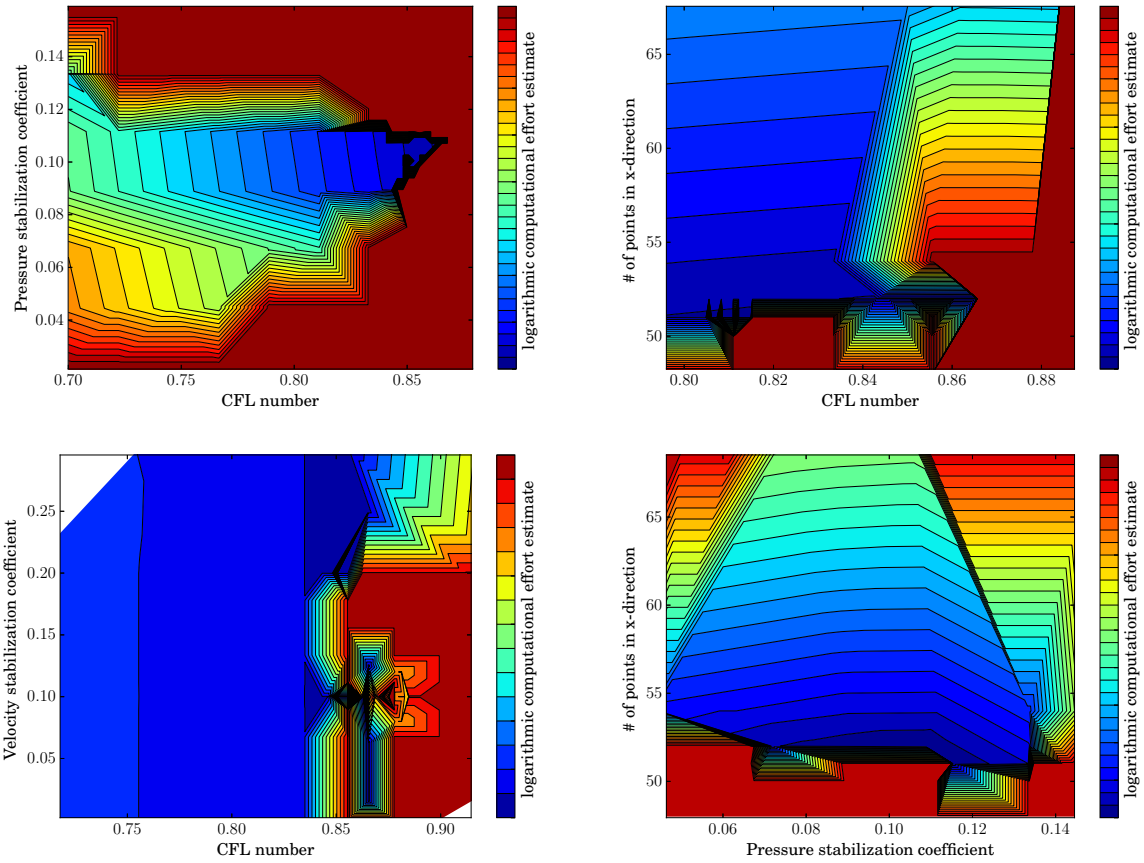


Figure A.1 Simulation parameter optimization for the duct flow at $\sqrt{10} \cdot 10^{-4}$ tolerance for the FD simulation.

A.2 Lid driven cavity

The full set of simulation parameters for the lid driven cavity at a target tolerance of 10^{-3} given in most pairwise combinations. An excerpt is used in Figure 5.8.



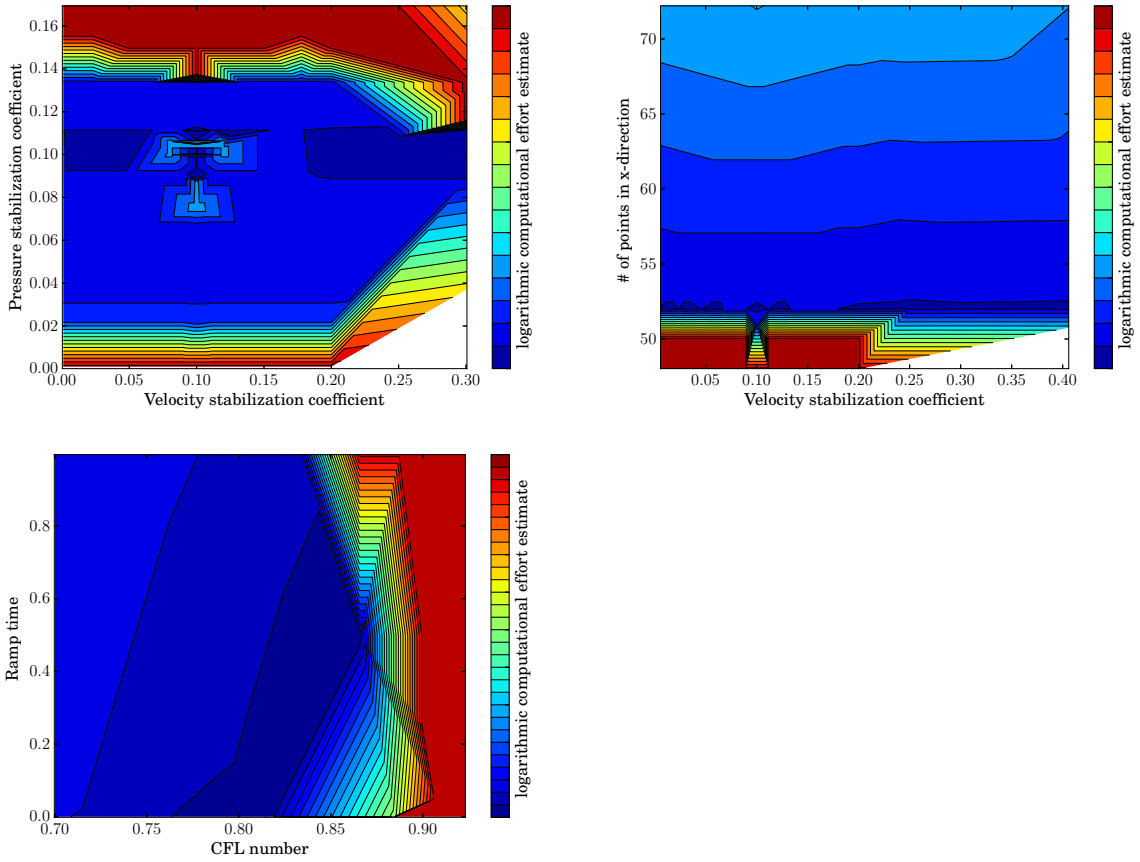
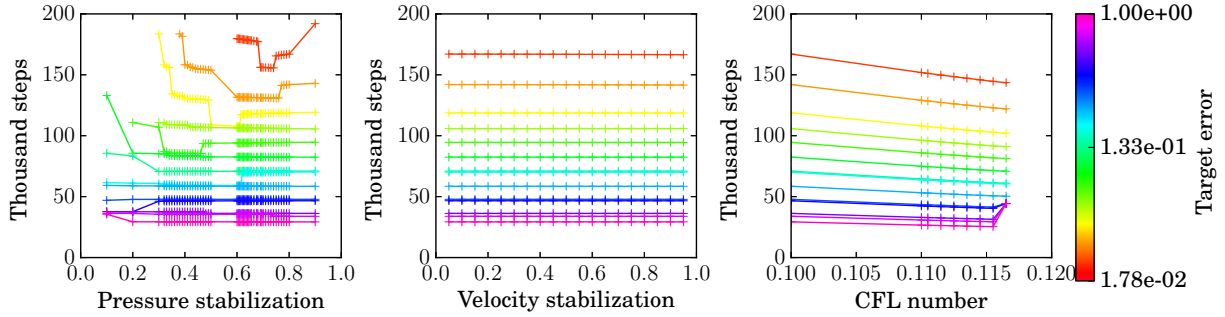


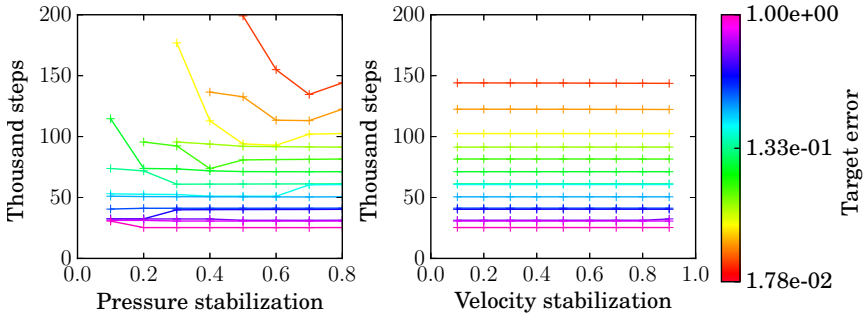
Figure A.2 Simulation parameter optimization for the ldc flow at 10^{-3} tolerance for the FD simulation.

A.3 Flow around a cylinder with a square cross section

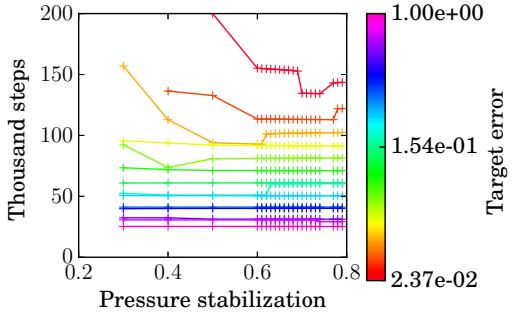
Simulation parameter influence and optimization sequence for the flow around a cylinder with a square cross section, using the FDM at a grid spacing of 0.005. The corresponding optimization sequence at a spacing of 0.01 can be found in Figure 5.16.



(a) First stage



(b) Second stage



(c) Third stage

Figure A.3 Maximum number of steps necessary to reach the relative target error for lift, drag and pressure difference. In each stage only the shown parameter was varied. The used grid spacing is 0.005.

Bibliography

[1] Top500 supercomputer sites. `https://www.top500.org/lists/2017/06/`, Aug. 2017.

[2] C. K. Aidun and J. R. Clausen. Lattice-Boltzmann method for complex flows. *Annual Review of Fluid Mechanics*, 42(1):439–472, 2010.

[3] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. *AFIPS Conference Proceedings*, 30:483–485, 1967.

[4] H. Aono, A. Gupta, D. Qi, and W. Shyy. The lattice Boltzmann method for flapping wing aerodynamics. *AIAA-2010-4867, 40th Fluid Dynamics Conference and Exhibit*, June 2010.

[5] P. Bailey, J. Myre, S. D. C. Walsh, D. J. Lilja, and M. O. Saar. Accelerating lattice Boltzmann fluid flow simulations using graphics processors. In *2009 International Conference on Parallel Processing*, pages 550–557, Sept 2009.

[6] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 2000.

[7] J. Baum, H. Luo, R. Löhner, E. Goldberg, and A. Feldhun. Application of unstructured adaptive moving body methodology to the simulation of fuel tank separation from an F-16 fighter. In *35th Aerospace Sciences Meeting and Exhibit, AIAA*, Jan. 1997.

[8] J. Bernsdorf, F. Durst, and M. Schäfer. Comparison of cellular automata and finite volume techniques for simulation of incompressible flows in complex geometries. *International Journal for Numerical Methods in Fluids*, 29(3):251–264, 1999.

[9] 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*, 94:511–525, May 1954.

[10] H. Bijl, D. Lucor, S. Mishra, and C. Schwab, editors. *Uncertainty Quantification in Computational Fluid Dynamics*. Springer International Publishing, Cham, 2013.

[11] W. J. Bolosky and M. L. Scott. False sharing and its effect on shared memory performance. In *USENIX Systems on USENIX Experiences with Distributed and Multiprocessor Systems - Volume 4*, Sedms'93, Berkeley, CA, USA, 1993. USENIX Association.

[12] S. Borkar. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

[13] M. Breuer, J. Bernsdorf, T. Zeiser, and F. Durst. Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-Boltzmann and finite-volume. *International Journal of Heat and Fluid Flow*, 21(2):186 – 196, 2000.

[14] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High Performance Computing Applications*, 14(3):189–204, 2000.

[15] J. M. Buick and C. A. Greated. Gravity in a lattice Boltzmann model. *Physical Review E*, 61:5307–5320, May 2000.

[16] D. Callahan, J. Cocke, and K. Kennedy. Estimating interlock and improving balance for pipelined architectures. *Journal of Parallel and Distributed Computing*, 5(4):334–358, Aug. 1988.

[17] M. H. Carpenter and C. A. Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. Technical Report NASA-TM-109112, NASA Langley Research Center, June 1994.

[18] S. Carr and K. Kennedy. Improving the ratio of memory operations to floating-point operations in loops. *ACM Transactions on Programming Languages and Systems*, 16(6):1768–1810, Nov. 1994.

[19] C. Cercignani. *The Boltzmann Equation and Its Applications*, volume 67 of *Applied Mathematical Sciences*. Springer New York, 1988.

[20] C. Cercignani. *Mathematical Methods in Kinetic Theory*. Springer US, 2 edition, 1990.

[21] B. Chapman, F. Bregier, A. Patil, and A. Prabhakar. Achieving performance under OpenMP on ccNUMA and software distributed shared memory systems. *Concurrency and Computation: Practice and Experience*, 14(8-9):713–739, 2002.

[22] S. Chen and G. D. Doolen. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998.

[23] T. P. Chiang, W. H. Sheu, and R. R. Hwang. Effect of Reynolds number on the eddy structure in a lid-driven cavity. *International Journal for Numerical Methods in Fluids*, 26(5):557–579, 1998.

[24] B. Chopard and A. Dupuis. A mass conserving boundary condition for lattice Boltzmann models. *International Journal of Modern Physics B*, 17:103–107, 2003.

[25] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 1967.

[26] A. Coppel, T. N. Gardner, N. Caplan, and D. M. Hargreaves. Simulating the fluid dynamic behaviour of oar blades in competition rowing. *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology*, 224(1):25–35, 2010.

[27] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Review*, 51(1):129–159, 2009.

[28] R. de la Cruz and M. Araya-Polo. Algorithm 942: Semi-stencil. *ACM Transactions on Mathematical Software*, 40(3):23:1–23:39, Apr. 2014.

[29] B. de Saint-Venant. Mémoire sur la dynamique des fluides. *Comptes Rendus de l'Académie des Sciences*, 17:1240–1242, 1843.

[30] L. Djoudi, D. Barthou, P. Carribault, C. Lemuet, J.-T. Acquaviva, and W. Jalby. MAQAO: Modular assembler quality analyzer and optimizer for Itanium 2. In *Workshop on Explicitly Parallel Instruction Computing Techniques*, Mar. 2005.

[31] B. Duncan, A. Fischer, and S. Kandasamy. Validation of lattice-Boltzmann aerodynamics simulation for vehicle lift prediction. *FEDSM-ICNMM 2010-30891*, 2010.

[32] Exa Corporation. PowerFLOW. `http://exa.com/en/product/simulation-tools/powerflow-cfd-simulation`, Aug. 2017.

[33] A. Fog. Instruction tables. `http://www.agner.org/optimize/instruction_tables.pdf`, Dec. 2016.

[34] U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J.-P. Rivet. Lattice gas hydrodynamics in two and three dimensions. *Complex Systems*, 1(4):649–707, 1987.

[35] K. Fujii. Progress and future prospects of CFD in aerospace—Wind tunnel and beyond. *Progress in Aerospace Sciences*, 41(6):455 – 470, 2005.

[36] D. J. Fyfe. Economical evaluation of Runge-Kutta formulae. *Mathematics of Computation*, 20(95):392–398, 1966.

[37] S. Geller, M. Krafczyk, J. Tölke, S. Turek, and J. Hron. Benchmark computations based on lattice-Boltzmann, finite element and finite volume methods for laminar flows. *Computers & Fluids*, 35(8–9):888 – 897, 2006.

[38] U. Ghia, K. Ghia, and C. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387 – 411, 1982.

[39] I. Ginzburg. Lattice Boltzmann modeling with discontinuous collision components. hydrodynamic and advection-diffusion equations. *Journal of Statistical Physics.*, (126):157–203, 2007.

[40] I. Ginzburg and D. d'Humières. Multireflection boundary conditions for lattice Boltzmann models. *Physical Review E*, 68:066614, Dec 2003.

[41] 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*, 3(3):519–581, 2008.

[42] 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*, 3(2):427–478, 2008.

[43] T. Glatzel, C. Litterst, C. Cupelli, T. Lindemann, C. Moosmann, R. Niekrawietz, W. Streule, R. Zengerle, and P. Koltay. Computational fluid dynamics (CFD) software tools for microfluidic applications – A case study. *Computers & Fluids*, 37(3):218 – 235, 2008.

[44] L. F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. PhD thesis, New Haven, CT, USA, 1987. AAI8727216.

[45] Z. Guo, C. Zheng, and B. Shi. Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Physical Review E*, 65:046308, Apr 2002.

[46] B. Gustafsson. The convergence rate for difference approximations to mixed initial boundary value problems. *Mathematics of Computation*, 29(130):396–406, 1975.

[47] B. Gustafsson. The convergence rate for difference approximations to general mixed initial-boundary value problems. *SIAM Journal on Numerical Analysis*, 18(2):179–190, 1981.

[48] G. Guzel and I. Koc. Time-accurate flow simulations using a finite-volume based lattice Boltzmann flow solver with dual time stepping scheme. *International Journal of Computational Methods*, 13(06):1650035, 2016.

[49] G. Hager, J. Treibig, J. Habich, and G. Wellein. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience*, 28(2):189–210, 2016.

[50] G. Hager and G. Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 2011.

[51] R. K. Hanna. CFD in sport - a retrospective; 1992 - 2012. *Procedia Engineering*, 34:622 – 627, 2012.

[52] X. He, G. D. Doolen, and T. Clark. Comparison of the lattice Boltzmann method and the artificial compressibility method for Navier–Stokes equations. *Journal of Computational Physics*, 179(2):439 – 451, 2002.

[53] X. He, Q. Zou, L.-S. Luo, and M. Dembo. Analytic solutions of simple flows and analysis of nonslip boundary conditions for the lattice Boltzmann BGK model. *Journal of Statistical Physics*, 87(1):115–136, Apr 1997.

[54] M. Hecht and J. Harting. Implementation of on-site velocity boundary conditions for D3Q19 lattice Boltzmann simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(01):P01018, 2010.

[55] F. Higuera and J. Jiménez. Boltzmann approach to lattice gas simulations. *Europhysics Letters*, 9(7), 1989.

[56] J. Hofmann and D. Fey. An ECM-based energy-efficiency optimization approach for bandwidth-limited streaming kernels on recent Intel Xeon processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, pages 31–38, Piscataway, NJ, USA, 2016. IEEE Press.

[57] J. Hofmann, D. Fey, M. Riedmann, J. Eitzinger, G. Hager, and G. Wellein. *Performance Analysis of the Kahan-Enhanced Scalar Product on Current Multicore Processors*, pages 63–73. Springer International Publishing, Cham, 2016.

[58] J. Hofmann, G. Hager, G. Wellein, and D. Fey. An analysis of core- and chip-level architectural features in four generations of Intel server processors. In J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, editors, *High Performance Computing: 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18–22, 2017, Proceedings*, pages 294–314, Cham, 2017. Springer International Publishing.

[59] H. Höller, A. Koskela, E. Dorfi, and W. Benger. Differential geometrically consistent artificial viscosity in comoving curvilinear coordinates. *ArXiv e-prints*, Apr. 2013.

[60] T. Inamuro, M. Yoshino, and F. Ogino. A non-slip boundary condition for lattice Boltzmann simulations. *Physics of Fluids*, 7(12):2928–2930, 1995.

[61] T. Inamuro, M. Yoshino, and F. Ogino. Erratum: "a non-slip boundary condition for lattice Boltzmann simulations" [phys. fluids 7, 2928 (1995)]. *Physics of Fluids*, 8(4):1124–1124, 1996.

[62] T. Inamuro, M. Yoshino, and F. Ogino. Accuracy of the lattice Boltzmann method for small Knudsen number with finite Reynolds number. *Physics of Fluids*, 9(11):3535–3542, 1997.

[63] Intel. 3rd generation Intel Core processor family quad core launch product information. `http://download.intel.com/newsroom/kits/core/3rdgen/pdfs/3rd_Generation_Intel_Core_Product_Information.pdf`, Apr. 2012.

[64] Intel. Intel Xeon processor E5 and E7 v3 family uncore performance monitoring reference manual. `https://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-v3-uncore-performance-monitoring.html`, June 2015.

[65] Intel. Intel 64 and IA-32 architectures optimization reference manual. `http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html`, June 2016.

[66] Intel. Intel architecture code analyzer, version 2.1, 2016.

[67] Intel. Intel C++ compiler 17.0 developer guide and reference. `https://software.intel.com/en-us/download/intel-c-compiler-170-developer-guide-and-reference`, 2016.

[68] Intel. Intel Xeon processor E5-2690 v4 (35M cache, 2.60 GHz) specifications. `http://ark.intel.com/products/91770/Intel-Xeon-Processor-E5-2690-v4-35M-Cache-2_60-GHz`, Nov. 2016.

[69] Intel. Intel intrinsics guide, version 2.8. `https://software.intel.com/en-us/comment/1758892`, Mar. 2017.

[70] Intel. Intel Xeon platinum 8180m processor (38.5m cache, 2.50 GHz) product specifications. `https://ark.intel.com/products/120498/Intel-Xeon-Platinum-8180M-Processor-38_5M-Cache-2_50-GHz`, Sept. 2017.

[71] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes. In *14th Fluid and Plasma Dynamics Conference*, Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics, June 1981.

[72] M. John D. Sustainable memory bandwidth in high performance computers. Technical report, University of Virgina, 2007.

[73] M. Junk. A finite difference interpretation of the lattice Boltzmann method. *Numerical Methods for Partial Differential Equations*, 17(4):383–402, 2001.

[74] M. Junk and A. Klar. Discretizations for the incompressible Navier–Stokes equations based on the lattice Boltzmann method. *SIAM Journal on Scientific Computing*, 22(1):1–19, 2000.

[75] M. Junk, A. Klar, and L.-S. Luo. Asymptotic analysis of the lattice Boltzmann equation. *Journal of Computational Physics*, 210(2):676–704, 2005.

[76] M. Junk and Z. Yang. Asymptotic analysis of lattice Boltzmann boundary conditions. *Journal of Statistical Physics*, 121(1):3–35, 2005.

[77] M. Junk and Z. Yang. Convergence of lattice Boltzmann methods for Navier-Stokes flows in periodic and bounded domains. *Numerische Mathematik*, 112(1):65–87, 2009.

[78] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams. An auto-tuning framework for parallel multicore stencil computations. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[79] D. Kandhai, D.-E. Vidal, A. Hoekstra, H. Hoefsloot, P. Iedema, and P. Sloot. Lattice-Boltzmann and finite element simulations of fluid flow in a SMRX Static Mixer Reactor. *International Journal for Numerical Methods in Fluids*, 31(6):1019–1033, 1999.

[80] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97(2):414 – 443, 1991.

[81] J. Kim, P. Moin, and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics*, 177:133–166, 1987.

[82] C. Kiris and D. Kwak. Numerical solution of incompressible Navier–Stokes equations using a fractional-step approach. *Computers & Fluids*, 30(7):829 – 851, 2001.

[83] B. Klein, F. Kummer, M. Keil, and M. Oberlack. An extension of the SIMPLE based discontinuous Galerkin solver to unsteady incompressible flows. *International Journal for Numerical Methods in Fluids*, 77(10):571–589, 2015.

[84] C. Knisely. Strouhal numbers of rectangular cylinders at incidence: A review and new data. *Journal of Fluids and Structures*, 4(4):371 – 393, 1990.

[85] J. M. V. A. Koelman. A simple lattice Boltzmann scheme for Navier-Stokes fluid flow. *Europhysics Letters*, 15(6):603–607, 1991.

[86] L. I. G. Kovasznay. Laminar flow behind a two-dimensional grid. *Mathematical Proceedings of the Cambridge Philosophical Society*, 44(01):58–62, 1948.

[87] M. Krafczyk, M. Cerrolaza, M. Schulz, and E. Rank. Analysis of 3D transient blood flow passing through an artificial aortic valve by lattice Boltzmann methods. *Journal of Biomechanics*, 1998.

[88] S. Kronawitter, H. Stengel, G. Hager, and C. Lengauer. Domain-specific optimization of two Jacobi smoother kernels and their evaluation in the ECM performance model. *Parallel Processing Letters*, 24(03):1441004, 2014.

[89] M. Kronbichler, A. Diagne, and H. Holmgren. A fast massively parallel two-phase flow solver for microfluidic chip simulation. *The International Journal of High Performance Computing Applications*, pages 1–22, 2016.

[90] D. Kwak, J. L. C. Chang, S. P. Shanks, and S. R. Chakravarthy. A three-dimensional incompressible Navier-Stokes flow solver using primitive variables. *AIAA Journal*, 24(3):390–396, Mar. 1986.

[91] A. J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, 271:285–309, 1994.

[92] M. D. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. *SIGPLAN Notices*, 26(4):63–74, Apr. 1991.

[93] A. Lapidus. A detached shock calculation by second-order finite differences. *Journal of Computational Physics*, 2(2):154 – 177, 1967.

[94] J. Latt, B. Chopard, O. Malaspinas, M. Deville, and A. Michler. Straight velocity boundaries in the lattice Boltzmann method. *Physical Reviews E*, 77:056703, May 2008.

[95] D. Lockard, L.-S. Luo, and B. Singer. Evaluation of the lattice-Boltzmann equation solver PowerFLOW for aerodynamic applications. *NASA/CR-2000-210550 ICASE Report No. 2000-40*, 2000.

[96] R. Löhner, A. T. Corrigan, K.-R. Wichmann, and W. A. Wall. Comparison of lattice-Boltzmann and finite difference solvers. In *52nd Aerospace Sciences Meeting*, AIAA SciTech. American Institute of Aeronautics and Astronautics, Jan. 2014.

[97] R. Löhner, K. Morgan, and J. Peraire. A simple extension to multidimensional problems of the artificial viscosity due to Lapidus. *Communications in Applied Numerical Methods*, 1(4):141–147, 1985.

[98] P. Louda, K. Korzel, and J. Příhoda. On using artificial compressibility method for solving turbulent flows. In J. Brandts, J. Chleboun, S. Korotov, K. Segeth, J. Šístek, and T. Vejchodský, editors, *Proceedings of the International Conference Applications of Mathematics 2012*, pages 163–172, 2012.

[99] D. Lucor, D. Xiu, C.-H. Su, and G. E. Karniadakis. Predictability and uncertainty in CFD. *International Journal for Numerical Methods in Fluids*, 43(5):483–505, 2003.

[100] R. Löhner, J. Cebral, O. Soto, P. Yim, and J. E. Burgess. Applications of patient-specific CFD in medicine and life sciences. *International Journal for Numerical Methods in Fluids*, 43(6-7):637–650, 2003.

[101] N. R. Mahapatra and B. Venkatrao. The processor-memory bottleneck: Problems and solutions. *Crossroads*, 5(3es):1–8, Apr. 1999.

[102] S. Marié, D. Ricot, and P. Sagaut. Comparison between lattice Boltzmann method and Navier-Stokes high order schemes for computational aeroacoustics. *Journal of Computational Physics*, 228:1056–1070, Mar. 2009.

[103] I. L. Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147–154, 2014.

[104] G. McNamara and G. Zanetti. Use of the Boltzmann equation to simulate lattice-gas automata. *Physical Review Letters*, 61:2332–2335, 1988.

[105] C. Merkle. Time-accurate unsteady incompressible flow algorithms based on artificial compressibility. In *8th Computational Fluid Dynamics Conference*, pages 397–407, Honolulu HI, 1987. AIAA.

[106] A. Mohamad and A. Kuzmin. A critical evaluation of force term in lattice Boltzmann method, natural convection problem. *International Journal of Heat and Mass Transfer*, 53(5):990 – 996, 2010.

[107] R. D. Moser, J. Kim, and N. N. Mansour. Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau = 590$. *Physics of Fluids*, 11(4):943–945, 1999.

[108] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[109] C. L. M. H. Navier. Mémoire sur les lois du mouvement des fluides. *Mémoires de l'Académie des Sciences*, 6:389–416, 1823.

[110] P. A. Newman, G. J. Hou, and A. C. Taylor. Observations regarding use of advanced CFD analysis, sensitivity analysis, and design codes in MDO. Technical report, Institute for Computer Applications in Science and Engineering Hampton VA, Mar. 1996.

[111] D. R. Noble, S. Chen, J. G. Georgiadis, and R. O. Buckius. A consistent hydrodynamic boundary condition for the lattice Boltzmann method. *Physics of Fluids*, 7(1):203–209, 1995.

[112] D. R. Noble, J. G. Georgiadis, and R. O. Buckius. Direct assessment of lattice Boltzmann hydrodynamics and boundary conditions for recirculating flows. *Journal of Statistical Physics*, 81:17–33, 1995.

[113] D. R. Noble, J. G. Georgiadis, and R. O. Buckius. Comparison of accuracy and performance for lattice Boltzmann and finite difference simulations of steady viscous flow. *International Journal for Numerical Methods in Fluids*, 23(1):1–18, 1996.

[114] T. Ohwada, P. Asinari, and D. Yabusaki. Artificial compressibility method and lattice Boltzmann method: Similarities and differences. *Computers & Mathematics with Applications*, 61(12):3461 – 3474, 2011. Mesoscopic Methods for Engineering and Science — Proceedings of ICMMES-09.

[115] A. Okajima. Strouhal numbers of rectangular cylinders. *Journal of Fluid Mechanics*, 123:379–398, 1982.

[116] OpenMP Architecture Review Board. OpenMP application programming interface version 4.5. `http://www.openmp.org/specifications`, Nov. 2015.

[117] S. A. Orszag, M. Israeli, and M. O. Deville. Boundary conditions for incompressible flows. *Journal of Scientific Computing*, 1(1):75–111, Mar 1986.

[118] C. Pan, L.-S. Luo, and C. T. Miller. An evaluation of lattice Boltzmann schemes for porous medium flow simulation. *Computers & Fluids*, 35(8–9):898 – 909, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[119] P. R. Panda, H. Nakamura, N. D. Dutt, and A. Nicolau. Augmenting loop tiling with data alignment for improved cache performance. *IEEE Transactions on Computers*, 48(2):142–149, Feb 1999.

[120] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. Morgan Kaufmann, Burlington, 4th edition, 2009.

[121] T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser. Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures. *Proceedings of the IEEE/ACM SC2004 Conference*, 2004.

[122] S. D. Poisson. Mémoire sur les équations générales de l'équilibre et du mouvement des corps solides élastiques et des fluides. *Journal de l'Ecole Polytechnique de Paris*, 13:139–166, 1831.

[123] Y. H. Qian, D. D'Humières, and P. Lallemand. Lattice BGK models for Navier-Stokes equation. *Europhysics Letters*, 17(6):479, 1992.

[124] G. Rivera and C.-W. Tseng. Tiling optimizations for 3D scientific computations. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 32–32, Nov 2000.

[125] S. E. Rogers and D. Kwak. Numerical solution of the incompressible Navier-Stokes equations for steady-state and time-dependent problems. In *27th Aerospace Sciences Meeting*, Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 1989.

[126] S. E. Rogers and D. Kwak. Upwind differencing scheme for the time-accurate incompressible Navier-Stokes equations. *AIAA Journal*, 28(2):253–262, Feb. 1990.

[127] C. J. Roth, L. Yoshihara, M. Ismail, and W. A. Wall. Computational modelling of the respiratory system: Discussion of coupled modelling approaches and two recent extensions. *Computer Methods in Applied Mechanics and Engineering*, 314(Supplement C):473 – 493, 2017.

[128] A. Saha, G. Biswas, and K. Muralidhar. Three-dimensional study of flow past a square cylinder at low Reynolds numbers. *International Journal of Heat and Fluid Flow*, 24(1):54 – 66, 2003.

[129] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher. *Benchmark Computations of Laminar Flow Around a Cylinder*, pages 547 – 566. Vieweg+Teubner Verlag, Wiesbaden, 1996.

[130] W. Schönauer. *Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers*. 2000.

[131] F. Schwabl. Basic principles. In *Statistical Mechanics*, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[132] S. Sen, S. Mittal, and G. Biswas. Flow past a square cylinder at low Reynolds numbers. *International Journal for Numerical Methods in Fluids*, 67(9):1160–1174, 2011.

[133] X. Shan and G. Doolen. Multicomponent lattice-Boltzmann model with interparticle interaction. *Journal of Statistical Physics*, 81(1):379–393, Oct 1995.

[134] P. N. Shankar and M. D. Deshpande. Fluid mechanics in the driven cavity. *Annual Review of Fluid Mechanics*, 32(1):93–136, 2000.

[135] A. J. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, Sept. 1982.

[136] S. Smolentsev, S. Badia, R. Bhattacharyay, L. Bühler, L. Chen, Q. Huang, H.-G. Jin, D. Krasnov, D.-W. Lee, E. M. de les Valls, C. Mistrangelo, R. Munipalli, M.-J. Ni, D. Pashkevich, A. Patel, G. Pulugundla, P. Satyamurthy, A. Snegirev, V. Sviridov, P. Swain, T. Zhou, and O. Zikanov. An approach to verification and validation of mhd codes for fusion applications. *Fusion Engineering and Design*, 100(C):65–72, 2015.

[137] A. Sohankar, C. Norberg, and L. Davidson. Simulation of three-dimensional flow around a square cylinder at moderate Reynolds numbers. *Physics of Fluids*, 11(2):288–306, 1999.

[138] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. *SIGPLAN Notices*, 34(5):215–228, May 1999.

[139] P. Spalart. Strategies for turbulence modelling and simulations. *International Journal of Heat and Fluid Flow*, 21(3):252 – 263, 2000.

[140] P. R. Spalart and V. Venkatakrishnan. On the role and challenges of CFD in the aerospace industry. *The Aeronautical Journal*, 120(1223):209–232, 2016.

[141] J. L. Steger and P. Kutler. Implicit finite-difference procedures for the computation of vortex wakes. *AIAA Journal*, 15(4):581–590, Apr. 1977.

[142] 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*, ICS '15, pages 207–216, New York, NY, USA, 2015. ACM.

[143] J. Steppeler, R. Hess, U. Schättler, and L. Bonaventura. Review of numerical methods for nonhydrostatic weather prediction models. *Meteorology and Atmospheric Physics*, 82(1):287–301, Jan 2003.

[144] G. G. Stokes. On the theories of the internal friction of fluids in motion, and of the equilibrium and motion of elastic solids. *Transactions of the Cambridge Philosophical Society*, 8:287–305, 1845.

[145] S. Succi. The lattice Boltzmann equation – for fluid dynamics and beyond. *Clarendon Pres*, 2001.

[146] J. Treibig and G. Hager. *Parallel Processing and Applied Mathematics: 8th International Conference, PPAM 2009, Wroclaw, Poland, September 13-16, 2009. Revised Selected Papers, Part I*, chapter Introducing a Performance Model for Bandwidth-Limited Loop Kernels, pages 615–624. Springer Berlin Heidelberg, 2010.

[147] 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*, pages 207–216, San Diego CA, Sept 2010.

[148] L. Verlet. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159:98–103, 1967.

[149] M. Vines, W.-S. Lee, and C. Mavriplis. Computer animation challenges for computational fluid dynamics. *International Journal of Computational Fluid Dynamics*, 26(6-8):407–434, 2012.

[150] J. von Neumann. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[151] I. Wald. Fast construction of SAH BVHs on the Intel many integrated core (MIC) architecture. *IEEE Transactions on Visualization & Computer Graphics*, 18:47–57, 2010.

[152] M. Weimer, M. Meinke, and E. Krause. *Numerical Simulation of Incompressible Flows with the Method of Artificial Compressibility*, pages 355–368. Vieweg+Teubner Verlag, Wiesbaden, 1996.

[153] P. Welander. On the temperature jump in a rarefied gas. *Arkiv för Fysik*, 7(44):507–553, 1954.

[154] P. Wesseling. *Principles of Computational Fluid Dynamics*, volume 29 of *Springer Series in Computational Mathematics*. Springer-Verlag Berlin Heidelberg, 1 edition, 2001.

[155] R. C. Whaley and A. M. Castaldo. Achieving accurate and context-sensitive timing for code optimization. *Software: Practice and Experience*, 38(15):1621–1642, 2008.

[156] F. White. *Viscous Fluid Flow*. McGraw-Hill international editions: Mechanical engineering series. McGraw-Hill, 1991.

[157] K.-R. Wichmann, M. Kronbichler, R. Löhner, and W. A. Wall. Practical applicability of optimizations and performance models to complex stencil-based loop kernels in CFD. *The International Journal of High Performance Computing Applications*, 2018.

[158] K.-R. Wichmann, M. Kronbichler, R. Löhner, and W. A. Wall. Towards a very efficient incompressible flow solver: Comparison of optimized lattice Boltzmann and finite difference solvers. submitted.

[159] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice Boltzmann simulation optimization on leading multicore platforms. *Proceedings of the 22nd IEEE International Symposium on Parallel & Distributed Processing*, 2008.

[160] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Optimization of a lattice Boltzmann computation on state-of-the-art multicore platforms. *Journal of Parallel and Distributed Computing*, 2009.

[161] S. W. Williams, A. Waterman, and D. A. Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Technical Report UCB/EECS-2008-134, EECS Department, University of California, Berkeley, Oct 2008.

[162] J. Williamson. Low-storage Runge-Kutta schemes. *Journal of Computational Physics*, 35(1):48 – 56, 1980.

[163] M. Wittmann, T. Zeiser, G. Hager, and G. Wellein. Comparison of different propagation steps for lattice Boltzmann methods. *Computers & Mathematics with Applications*, 65(6):924 – 935, 2013. Mesoscopic Methods in Engineering and Science.

[164] D. A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*, volume 1725 of *Lecture Notes in Mathematics*. Springer-Verlag Berlin Heidelberg, 1 edition, 2000.

[165] M. Wolfe. More iteration space tiling. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, pages 655–664, New York, NY, USA, 1989. ACM.

[166] P. Woodward and P. Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54(1):115 – 173, 1984.

[167] M. Yoshino, Y. Matsuda, and C. Shao. Comparison of accuracy and efficiency between the lattice Boltzmann method and the finite difference method in viscous/thermal fluid flows. *International Journal of Computational Fluid Dynamics*, 18(4):333–345, 2004.

[168] C. Yount. Vector folding: Improving stencil performance via multi-dimensional SIMD-vector representation. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 865–870, Aug 2015.

[169] D. Yu, R. Mei, L.-S. Luo, and W. Shyy. Viscous flow computations with the method of lattice Boltzmann equation. *Progress in Aerospace Sciences*, 39(5):329 – 367, 2003.

[170] T. Zeiser, G. Hager, and G. Wellein. Benchmark analysis and application results for lattice Boltzmann simulations on NEC SX vector and Intel Nehalem systems. *Parallel Processing Letters*, 19:491–511, 12 2009.

[171] T. Zeiser, G. Hager, and G. Wellein. Vector Computers in a World of Commodity Clusters, Massively Parallel Systems and Many-Core Many-Threaded CPUs: Recent Experience Based on an Advanced Lattice Boltzmann Flow Solver. In *High Performance Computing in Science and Engineering '08: Transactions of the High Performance Computing Center, Stuttgart (HLRS) 2008*, volume 5 of *Mathematics and Statistics*, pages 333–347. Springer, Berlin Heidelberg, 2009.

[172] Q. Zou and X. He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, 9(6):1591–1598, 1997.