Universität Stuttgart

Adaptive Grid Implementation for Parallel Continuum Mechanics Methods in Particle Simulations

Vom Stuttgarter Zentrum für Simulationswissenschaften (SC SimTech) und der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Michael Stefan Lahnert

aus Oberwesel

Hauptberichter:	Prof. Dr. Miriam Mehl
Mitberichter:	Prof. Dr. Carsten Burstedde
Mitprüfer:	Prof. Dr. Daniel Weiskopf

Tag der mündlichen Prüfung: October 14, 2019

Institute for Parallel and Distributed Systems

2019

Focusing your life solely on making a buck shows a poverty of ambition. It asks too little of yourself. And it will leave you unfulfilled.

— Barack Obama, 2006

Although this thesis was written with uttermost care, it cannot be ruled out that it contains errors. Please send any corrections and mistakes to michael.lahnert@gmail.com.

Contents

Lists of Figures, Tables, and Algorithms

List of Figures

List of Tables

List of Algorithms

List of Symbols and Acronyms

Symbol	Meaning	Page with First Occurrence
ρ	Fluid density	47
$\boldsymbol{p}$	Fluid momentum	47
p	Fluid pressure	44
$\overline{\pi}$	Fluid stress tensor	47
$\boldsymbol{u}$	Fluid velocity	44
Q	Volumetric flow rate	104
ζ	Fluid vorticity	98
ℓ_B	Bjerrum length $\ell_B = \frac{e^2}{4\pi\varepsilon k_B T}$	52
k_B	Boltzmann constant	44
e	Elementary charge	44
ε	Electric permittivity	44
$\boldsymbol{E}$	Electric field $\boldsymbol{E} = -\nabla\Phi$	57
Φ	Electrostatic potential	44
$\boldsymbol{f}$	Force	44
Γ	Friction coefficient	57
T	Temperature	44
ϱ	Net charge density $\varrho = \sum_\pm z_\pm e c_\pm$	52
c	Ionic species density	44
D	Diffusion coefficient	44
$\boldsymbol{j}$	Ionic flux	44
μ	Ionic species mobility	44
z	Ionic species valency	44
L_{ik}	Lattice-Boltzmann collision operator	46
f_i	Lattice-Boltzmann probability density in discrete direction i	46
f_i^{eq}	Lattice-Boltzmann equilibrium probability density in discrete direction i	47
c_s	Lattice-Boltzmann speed of sound	47
$\boldsymbol{c}_i$	Discrete direction of a given lattice-Boltzmann stencil	46
$\mathbb{N}$	Natural numbers without zero $(1, 2, 3, \ldots)$	30
$\mathbb{N}_0$	Natural numbers with zero $(\mathbb{N} \cup \{0\})$	84
$\mathbb{R}$	Real numbers	16

Symbol	Meaning	Page with First Occurrence
$\|\cdot\|_2$	ℓ_2 norm $\|\boldsymbol{x}\|_2 := \sqrt{\sum_{t=1}^{d} x_t^2}$	51
$\mathcal{O}(f(x))$	Big-$\mathcal{O}$ Landau notation	25
δ_{ik}	Kronecker delta $\delta_{ik} = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$	47
$\land$	Logical “and”	30
xor	Bitwise “exclusive or”	76
r_c	Cut-off radius of a MD potential	54
$\boldsymbol{v}$	Particle velocity	57
$\lvert\cdot\rvert!$	Number of elements of a set	74
AMR	Adaptive mesh refinement	26
BFS	Breadth-first search	36
CPU	Central processing unit	50
CSR	Compressed sparse row	37
DFS	Depth-first search	30
DFT	Discrete Fourier transform	46
DG	Discontinuous Galerkin	68
EK	Electrokinetic	44
FCT	Finest common tree	18
FDM	Finite difference method	24
FEM	Finite element method	24
FFT	Fast Fourier transform	53
FMA	Fused multiply-add	53
FVM	Finite volume method	24
GPL	GNU General Public License	37
GPU	Graphics processing unit	26
LBM	Lattice-Boltzmann method	24
LJ	Lennard-Jones	54
MD	Molecular dynamics	24
MRT	Multi-relaxation-time	47
SFC	Space-filling curve	26
SOR	Successive over-relaxation	46
SRT	Single-relaxation-time	47
TRT	Two-relaxation-time	47

Zusammenfassung

Die Arbeit befasst sich mit der minimal-invasiven Integration dynamisch-adaptiver baumstrukturierter Rechengitter in bestehende Anwendungen. Viele Simulationsprogramme beinhalten komplizierte naturwissenschaftliche Modelle und sind in vielen Personenjahren Entwicklungsarbeit entstanden. Zur räumlichen Diskretisierung werden allerdings einfache reguläre, kartesische Gitter benutzt. Bei der Untersuchung von Systemen, für deren Simulation ein größeres Volumen mit einer feinen Auflösung berechnet werden muss, stoßen diese Gitter an Grenzen, insbesondere hinsichtlich des Speicherverbrauchs.

Um die Berechnung größerer Gebiete und längerer Zeitskalen zu ermöglichen, sind räumlich-adaptive Gitter ein bekannter Lösungsweg. Werden strukturierte Gitter verwendet, können während der Laufzeit mit moderatem Aufwand lokale Gitteränderungen durchgeführt werden. Bei der Integration solcher Gitter in bestehende Anwendungen soll im Zuge dessen nicht das in der Anwendung vorhandene Expertenwissen wegen der neuen räumlichen Diskretisierung verloren gehen. Unser Ziel besteht also darin, räumliche Adaptivität in existierenden Simulationsprogrammen nachzurüsten und dabei möglichst große Teile des ursprünglichen Codes zu erhalten, um möglichst viel enthaltenes Wissen zu konservieren.

Wir verwenden die bekannte und skalierbare `p4est` Gitterbibliothek, um das Gitter darzustellen und von der Anwendung abzukapseln. Diese Bibliothek erweitern wir so, dass sie sich leichter in bestehende Anwendungen integrieren lässt. Wir demonstrieren unser Modell am Beispiel der Simulation von DNA-Translokation durch Nanoporen, modelliert mithilfe des Simulationscodes ESPResSo, der speziell für die Simulation von weicher Materie geeignet ist. Nach der Erweiterung von `p4est` sowie der Portierung der ersten physikalischen Komponente [1–3], konnten die übrigen physikalischen Subsysteme in zwei studentischen Masterarbeiten portiert werden [4, 5].

Die übrige Arbeit ist wie folgt gegliedert. Kapitel 2 erläutert verschiedene Formen räumlicher Adaptivität mit ihren jeweiligen Vorzügen und Defiziten vor. Wir entwickeln eine Schnittstelle für adaptive Gitter, um diese in bestehende Anwendungen zu integrieren und betrachten verschiedene Ansätze zur Implementierung dieser Schnittstelle. Außerdem

stellen wir die relevanten Teile der `p4est` Bibliothek vor und erläutern, wie die erwähnten Algorithmen hier umgesetzt wurden.

In Kapitel 3 geben wir einen detaillierten Überblick über das physikalische Modell und die ESPResSo Software. Wir stellen die Gleichungen der einzelnen physikalischen Subsysteme vor, aus denen sich die Simulation zusammensetzt, erläutern deren Kopplung sowie die in ESPResSo verwendete Diskretisierung.

Die Erweiterung der Diskretisierung für adaptive Gitter stellen wir in Kapitel 4 vor. Insbesondere erläutern wir den Umgang mit der zusätzlichen Komplexität aus Verfeinerungsgrenzen in der Diskretisierung durch virtuelle Zellen.

Die Beschreibung der Integration von ESPResSo und `p4est` folgt in Kapitel 5. Wir beschreiben die Vorarbeiten in `p4est` sowie die wichtigsten Schritte in ESPResSo. Wir erläutern unser Verfahren zur gemeinsamen Partitionierung mehrerer unabhängiger `p4est` Instanzen mithilfe des Finest-Common-Tree (FCT) und wie wir zu einer Position $x \in \Omega \subseteq \mathbb{R}^3$ die richtige Zelle in einer `p4est` Instanz finden; beispielsweise für die Kopplung oder um Partikel einer Zelle im Linked-Cell Verfahren zuzuweisen.

In Kapitel 6 demonstrieren wir die physikalische Korrektheit unserer Integration anhand bekannter Beispiele mit bekannter analytischer Lösung. Außerdem analysieren wir das Skalierungsverhalten und die Performance verschiedener Komponenten auf dem Tier 1 Supercomputer “Hazel Hen” am HLRS in Stuttgart. Zusätzlich geben wir einen kurzen Ausblick auf das Gesamtsystem mithilfe eines vorläufigen Modells der in [6] modellierten Porengeometrie, erweitert auf 3D. Hierbei verwenden wir einen größeren Porendurchmesser und verzichten auf Partikel.

Kapitel 7 fasst die Arbeit zusammen und zeigt Ansätze für weitergehende Arbeiten auf.

Abstract

This work presents a way for integrating dynamically-adaptive tree-structured grids into existing applications in a minimally invasive way. Multiple simulation software feature complex scientific models and have been developed in several man-years. Often, their spatial discretization uses simple regular Cartesian grids. If we study systems that require simulating larger volumes with fine grid resolution, these grids reach their limits, especially regarding memory-consumption.

To allow simulating systems on larger time and length scales, spatially-adaptive grids are a well-known way to mitigate this issue. If we use structured adaptive grids, we can change the discretization during run time at moderate cost. At the same time, we do not want to lose the existing domain-specific knowledge contained in the application for no other reason that changing the spatial discretization. Thus, we aim for integrating dynamic spatial adaptivity into legacy codes while preserving most of the original knowledge contained in the application.

We use the well-known and scalable `p4est` grid library to represent the grid and encapsulate it from the application. We add features to this library to facilitate its integration into legacy codes. We demonstrate our model by simulating DNA translocation through nanopores using the soft matter simulation software ESPResSo. After the initial extensions to `p4est` have been made and the first component was ported [1–3], porting the remaining subsystems could be done in two students Master's theses [4, 5].

The rest of the work is structured as follows. Chap. 2 introduces different ways for realizing spatial adaptivity with their respective benefits and deficits. We develop an interface for adaptive grids to integrate them into existing applications. We consider different algorithms to implement the respective parts of the interface. Additionally, we present the `p4est` library and describe its implementation of the interface.

In Chap. 3, we describe the underlying physical model of our system in detail. We present the system of equations for the individual physical subsystems that built our exemplary system and their coupling. Additionally, we describe the discretization used in ESPResSo.

We extend the discretization to adaptive grids in Chap. 4. We introduce virtual cells for dealing with the complexity of refinement boundaries in the spatial discretization.

Chap. 5 describes the integration of ESPResSo with `p4est`, focussing on the work in `p4est` and the main aspects in ESPResSo. We explain our algorithm for a joint partitioning of multiple independent `p4est` instances using the finest common tree (FCT) and how we identify the cell in a given `p4est` instance containing a position $x \in \Omega \subseteq \mathbb{R}^3$, e.g., for coupling or for inserting particles into cells of the grid of the Linked-Cell method.

We show results of the work in this thesis in Chap. 6. We verify the correctness of our implementation using systems whose physics are well-known and understood. Then, we go into detail about performance and scalability of individual subsystems and their coupling using the Tier 1 supercomputer “Hazel Hen” at the HLRS in Stuttgart. Additionally, we give a brief outlook on simulating the target system using a preliminary three-dimensional geometry derived from the pore geometry in [6] using a larger pore diameter.

Chap. 7 summarizes the work and gives a brief outlook.

Publications

During this PhD project, I created and contributed to a number of publications. Some parts of the work are already published there:

1. **M. Lahnert**; T. Aoki; C. Burstedde; M. Mehl. “Minimally-Invasive Integration of p4est in ESPResSo for Adaptive Lattice-Boltzmann.” *The 30th Computational Fluid Dynamics Symposium*. Japan Society of Fluid Mechanics, 2016. Entry [2] in bibliography.

2. **M. Lahnert**; C. Burstedde; C. Holm; M. Mehl; G. Rempfer; F. Weik. “Towards Lattice-Boltzmann on Dynamically Adaptive Grids – Minimally-Invasive Grid Exchange in ESPResSo.” *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*. Ed. by M. Papadrakakis; V. Papadopoulos; G. Stefanou; V. Plevris. ECCOMAS, 2016. Entry [1] in bibliography.

3. S. Hirschmann; M. Brunn; **M. Lahnert**; C. W. Glass; M. Mehl; D. Pflüger. “Load Balancing with p4est for Short-Range Molecular Dynamics with ESPResSo.” Ed. by S. Bassini; M. Danelutto; P. Dazzi; G. R. Joubert; F. Peters. Vol. 32. Advances in Parallel Computing. IOS Press, 2017, pp. 455–464. Entry [205] in bibliography.

4. S. Hirschmann; **M. Lahnert**; C. Schober; M. Brunn; M. Mehl; D. Pflüger. “Load-Balancing and Spatial Adaptivity for Coarse-Grained Molecular Dynamics Applications.” *High Performance Computing in Science and Engineering ’18*. Ed. by W. E. Nagel; D. H. Kröner; M. M. Resch. Springer International Publishing, 2018, pp. 409–423. Entry [207] in bibliography.

5. M. Mehl; **M. Lahnert**. “Adaptive grid implementation for parallel continuum mechanics methods in particle simulations.” *The European Physical Journal Special Topics* 227.14 (2019), pp. 1757–1778. Erratum. **M. Lahnert**; Burstedde; Mehl [6]. Entry [206] in bibliography.

6. **M. Lahnert**; C. Burstedde; M. Mehl. “Erratum to: Adaptive grid implementation for parallel continuum mechanics methods in particle simulations.” *European Physical Journal Special Topics* 227 (2019), pp. 1757–1778. Entry [226] in bibliography.

Preface

> *I’ve missed more than 9000 shots in my career. I’ve lost almost 300 games. 26 times, I’ve been trusted to take the game winning shot and missed. I’ve failed over and over and over again in my life. And that is why I succeed.*
>
> — Michael Jordan

This work would not have been possible without the help from lots of great people who supported me in lots of different ways. I want to thank

Prof. Dr. Miriam Mehl for giving me the opportunity to pursue this work. You challenged and supported me in more ways than I can imagine. While constantly offering guidance throughout my work, you gave me the freedom to follow projects according to my personal interests. I am very grateful for the countless hours you have spent reading short and not so short texts and text snippets or discussing the current state of my work.

Prof. Dr. Carsten Burstedde for your efforts in supervising me. You have taught me a lot, in hour-long sessions discussing `p4est` APIs in Stuttgart and Bonn as well as in discussing more mundane topics at conferences or in our free time.

Prof. Dr. Takayuki Aoki and his group at the Tokyo Institute of Technology for hosting me during my stay-abroad in Japan. Living in Japan and being part of your group was one of the greatest experiences I’ve ever had.

My collaborators for countless fun discussions about ESPResSo, physics, and more. You are: Carolin Schober and Steffen Hirschmann from SGS/SSE and Georg Rempfer, Florian Weik, Ingo Tischler, and Christian Holm from the ICP.

My coworkers for great discussions and fun at the ÖZ, the table football, during coffee breaks, and beyond. You are: Alexander, Amin, Benjamin, Carolin, David, Dirk, Fabian, Florian, Gregor, Julian, Kai, Klaudius, Kyle, Malte, Mario, Marvin, Michael, Nehzat, Raphael, Stefan, Steffen, Theresa, and anyone I forgot to mention here.
Special thanks to Julian for providing this awesome template.

My student coworkers Axel, Benjamin K., Benjamin M., Daniel, Felix, Ingo, and Malte.

The **Collaborative Research Center (SFB) 716** and the **Cluster of Excellence SimTech** for providing funding and getting to know lots of great people.

Anyone performing administrative tasks, thus, significantly relieving workloads and stress from PhD students such as myself at the IPVS, in SimTech, the SFB 716, and at Tokodai. You are: Andreas, Barbara, Bernd, Chieko-san, Christine, Claudia, Meike, Manfred, Ralf, and Stefanie.

My family and friends for always having my back, believing in me, and supporting me.

Stuttgart, August 28, 2019
Michael Stefan Lahnert

1 Introduction

Simulations play an increasingly important role in modern society. From developing new products, predicting the weather, or understanding processes at very large or very small scales; simulation has emerged as the third pillar of science besides theory and experiments for numerous reasons. These include, among others, that performing experiments may be expensive, difficult or even outright impossible. Simulations offer an alternative path for studying systems whose setup can, at least theoretically, be fully customized and whose properties can be easily investigated throughout the domain.

Relevant systems of interest typically combine different fundamental processes such as motion, flow, structural mechanics, or population growth. Each of these processes is typically described by a set of (partial) differential equations. When modeling a system, we must choose the relevant size of the system, the relevant processes steering it, define ways in which the chosen processes interact with each other, and define an initial state as well as boundary conditions to model the interactions of the system at hand and the outside world.

1.1 Simulations on Different Time and Length Scales

Setting up a simulation is challenging. If we want to study a certain effect, we have to model the system within a prescribed regime of time and length scales. We can model, e.g., flow phenomena at four different scales. To increase the scale from small to large, we reduce the level of detail. In turn, we can consider larger time and length scales. At the most detailed scale, the *quantum scale*, our system is governed by the Schrödinger equation. Our system representation includes the full electronic structure and position of nuclei including relativistic corrections and quantum effects [7, 8]. If we model individual atoms as spheres, we represent the system at the *microscopic scale*. The interactions of atoms is governed by Newton's equations of motion. All atomic interactions are visible and observable (molecular dynamics (MD)). Interaction potentials allow simpler modeling of the electronic structure [9]. Using a priori information about the modeled system at hand, we can locally bridge scales using the quantum mechanics molecular method (QMMM) [10] and locally include the electronic structure into the microscopic model. At the *mesoscopic scale*, we substitute groups of atoms by a probabilistic description using Boltzmann's equation that models the interaction of those super-particles. Thus, we can no longer observe atomic interactions. There are multiple approaches, e.g., dissipative particle dynamics (DPD) [11, 12], multi-particle collision dynamics (MPC) [13, 14], smoothed particle hydrodynamics (SPH) [15, 16], or the lattice-Boltzmann method (LBM) [17, 18]. At the *macroscopic scale*, we neglect individual elements forming the fluid altogether and describe the fluid from continuous properties such as density and flow velocity using the Navier-Stokes equations. The most widely used approaches for macroscopic fluid modeling are the finite difference method (FDM) [19, 20], the finite element method (FEM) [21, 22], and the finite volume method (FVM) [23, 24].

Getting the modeling of a system right is a hard problem. That is why Maître et al. consider it one of the three main sources of error in simulations besides numerical errors and data errors [25].

1.2 Target System: DNA Translocation Through a Nanopore

For the project at hand, we want to model DNA translocation through a nanopore. This target application was first described in [26]. We simulate a domain containing an ionic fluid and the DNA string. A membrane splits the domain into two dedicated basins connected through the thin channel of a nanopore. By applying a current, the DNA string should move through the pore in a controlled way. We sketch the setup in Fig. 1.1, [27, 28].

At the end of the day, the goal is to answer the question if molecules and colloidal particles can be characterized or manipulated using such a system. This would be a major break-through, as it would, e.g., allow producing personalized medicine [29]. First approaches to model this in a simulation using the ESPResSo software have been made in [30]. ESPResSo is a feature-rich simulation software [31–33] designed for soft matter applications. Other software packages in this field are, e.g., ESPResSo++ [34, 35], LAMMPS [36], GROMACS [37, 38], or ls1 mardyn [39–43].

This system is too large to be fully modeled at the microscopic scale. Instead, we want to model the DNA string microscopically and use a continuous model for the ionic

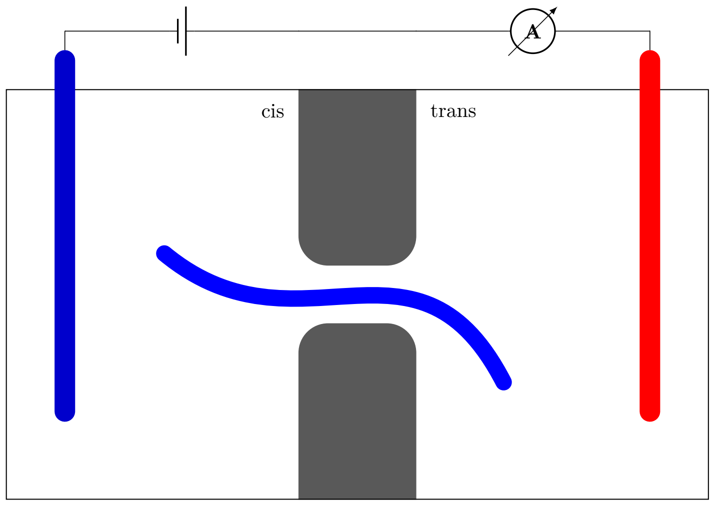


FIGURE 1.1 Our target simulation scenario: schematic setup of a DNA translocation experiment from [27, 28]. The basin is modeled as a rectangle, the gray blocks illustrate the nanopore and the curved blue line is a coarse-grained DNA model. An electric field is applied by the red and blue electrodes.

fluid and couple the respective subsystems. It turned out that coarse graining alone, i.e., fitted force-fields or boundary conditions are not enough to accurately model the system [27]. Therefore, together with our collaborators from the Physics department, we want to simulate the full system and see if this allows developing new boundary conditions. In Sec. 6.3, we present an exemplary configuration of the nanopore system with a total system size of 64 µm. Eventually, we want to simulate pore diameters in the range of 150 nm to 7.5 nm. Thus, we must target a grid spacing of 1 nm or less which is hard to achieve for regular Cartesian grids: in this system we would need at least $(\frac{64}{0.001})^3 = 2.6 \cdot 10^{14}$ cells, which induces memory requirements of $\mathcal{O}(1\,\text{PB})$. Among the first ten systems of the TOP500 list of the world's largest supercomputers[1] released in June 2019, only six systems can provide this amount of memory.

Additionally, it is clear that our simulations must be run in parallel. Not only does Moore's Law no longer apply to single node performance [44] but the sheer size of the system requires distributing the load among multiple processors. Domain decomposition approaches are the most commonly used approach for parallelizing simulations [45]. Here, each process is assigned a disjoint part Ω_i of the simulation domain Ω such that $\bigcup_i \Omega_i = \Omega$. To maximize the benefits of parallelization, we want to partition the system in such a way that for each process the computation takes the same amount of time. In this case, the load is evenly distributed among the processors or balanced.

Most modern-day simulation software is implemented as distributed-memory appli-

[1]`https://www.top500.org/lists/`

cations, relying on message-oriented communication using the message-passing interface (MPI) [46]. Some algorithms allow using graphic cards (graphics processing unit (GPU)) or other accelerator hardware using, e.g., NVIDIA's CUDA [47] language or some domain-specific language used for automatic code-generation [48]. In terms of Flynn's taxonomy, we are in an multiple instruction multiple data (MIMD) setting [49].

The goal of this project is simulating the full nanopore system using structured adaptive mesh refinement (AMR) and a newly developed coarse grain DNA model [50]. AMR is a well-known way to reduce the number of unknowns in a system while achieving good performance and parallel scalability [51, 52].

1.3 Integrating Tree-Structured Grids Into Legacy Codes

Changing fundamental aspects of legacy codes such as the spatial discretization is hard, because usually many dependencies have been implemented in the code over time. Moreover, modern simulation scenarios represent complex processes requiring different physical models to be mapped from the real-world application. In most cases, we have an established application code on the one hand and an optimized grid-library on the other hand. Both software packages have specific constraints to data-storage, data-access patterns, or communication schemes.

We propose our approach to answering the question of how both software packages can be integrated without rewriting the application from scratch. To integrate tree-structured grids into legacy codes minimally invasive, we suggest a three-step process [2]: First, replace the original grid with a regular tree-structured grid. The order of the numerical payload has to be adapted to match the space-filling curve (SFC) underlying the tree-structured grid. Additionally, partitioning the grid and exchanging data at process boundaries can be delegated to the grid library. This step must be used to verify that changing the traversal order of cells and the domain decomposition does not change numerical results. In the second step, we introduce actual spatial adaptivity and deal with numerical issues such as hanging nodes. In this work, we add virtual cells at refinement boundaries, i.e., we virtually overlay coarse cells at refinement boundaries with their children. This allows exchanging data logically with cells of the same size and introduces a local data-exchange between the levels by interpolation and restriction. In the final step, we want to make optimal use of the new grid implementation by thoroughly investigating the implementation for performance hotspots and implementing suitable optimizations. Examples are improved data-access patterns, implementing communication hiding, or whatever else seems appropriate to reduce the time to solution.

1.4 Contributions in This Work

In summary, the main contributions in this work are

- providing random-access to neighboring cells in `p4est` as required by most applications;
- integrating virtual cells in `p4est` to deal with refinement boundaries;
- an iterator that allows visiting specific cells, e.g., cells of a given refinement level, without traversing the entire grid;
- integrating ESPResSo with `p4est`, porting all relevant physical subsystems of our target application and their coupling scheme;
- extending their discretizations to dynamically-adaptive tree-structured grids; and
- partitioning multiple independent `p4est` instances using the finest common tree (FCT).

Compared to highly optimized software such as waLBerla [53–58] or the lattice-Boltzmann method (LBM) software developed in the group of Manfred Krafczyk [59, 60], which were specifically developed for performance, our approach is slower. Instead, we retain the domain-specific knowledge of the existing code. Both tasks, writing high-performance extreme-scale software and simulating multi-physics systems spanning several scales, are complex and often conflicting with each other.

For this reason, Maruyama et al. propose developing special purpose software that is tailored for specific applications [61]. They have ported the operative Japanese weather prediction model "ASUCA" to GPUs [48]. They obtained speed-ups of up to 5x while over 85% of the existing code base have been left untouched [61]. This, however, still involved changing more than 20,000 lines of code. We report on the amount of code we changed in Chap. 7.

We have taken a similar approach, although we have extended a generic grid library which is well-known and well-scaling. `p4est` is the grid library of the ACM Gordon Bell Prize winner in 2015 [52] and of Gordon Bell Prize finalists in 2008, 2010, and 2012 [62]. We develop an interface between existing software packages using regular Cartesian grids and `p4est` and test it for our target application ESPResSo.

2 Tree-Structured Cartesian Grids

In this chapter, we present our main reasons for using tree-structured Cartesian grids. Additionally, we define a high-level interface that a grid must provide to satisfy our requirements and present methods to implement it. Finally, we present the `p4est` grid library which we use and extend for our goal to integrate it into a legacy code in a minimally-invasive way.

2.1 Fundamental Ideas

In this section we give an overview on how to realize an adaptive spatial discretization. We derive a high-level interface to integrate a grid library with an application. We briefly summarize different ways for implementing sub-components of this interface.

2.1.1 Different Ways to Realize Grid-Adaptivity

There are several ways to discretize a spatial domain by an adaptive grid, which fall into two major categories: structured and unstructured grids. Albeit unstructured grids offer potentially perfect boundary approximation, they are not particularly well suited for dynamic simulations. In dynamic scenarios, regions-of-interest may move through the simulation domain, form, or vanish. Thus, dynamic simulation scenarios require constantly adapting the grid to the current state of the simulation. This leads to frequent re-meshing, a global operation for unstructured grids that cannot be readily afforded in highly parallel and highly dynamic scenarios. Additionally, unstructured grid usually come with a high memory consumption, because for each cell all neighbor relations across faces, edges, and corners must be stored.

Therefore, we will use structured grids. There are, however, several ways for constructing structured grids, e.g., moving meshes [63], patch-based grids [64], or tree-based

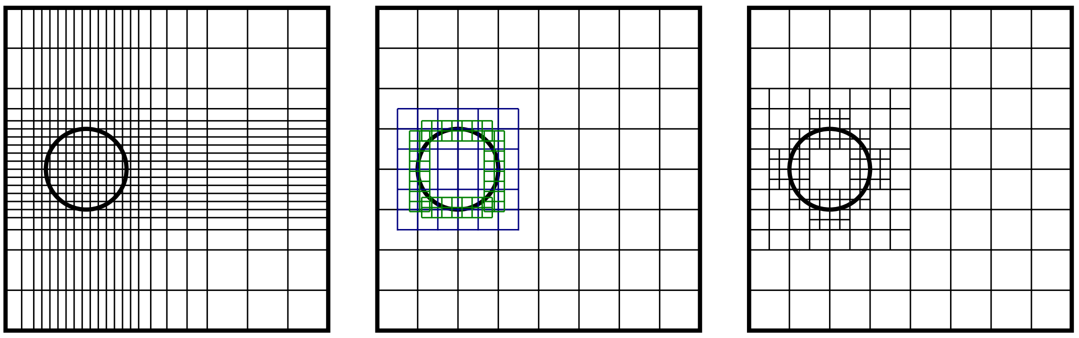


FIGURE 2.1 Schematic representation of different types of structured grid-adaptivity for resolving the geometry of a circle. From left to right: A moving mesh, patch-based AMR, and tree-based AMR.

grids [65]. Examples for each mentioned grid type discretizing the geometric boundary of a circle on a plane are shown in Fig. 2.1.

Several implementations of patch-based and tree-based grid-libraries have been compared in [66]. Additionally, Donna Calhoun maintains a collection of links related to adaptive mesh refinement (AMR) on her website[1].

In *patch-based* grids, adaptivity is realized as areas ("patches") of finer resolution, see Fig. 2.1. This leads to more cells than mathematically required and limits the discretization's flexibility. Examples for patch-based AMR-codes are, e.g., (i) AMROC [67], (ii) BoxLib [68–70] and its successor AMReX [71], (iii) Carpet [72, 73] as an AMR-framework for Cactus [74], (iv) Chombo [75], or (v) ENZO [76].

Unlike patch-based grids, *tree-based* grids are strictly coupled to a quadtree, octree, or a similar k-spacetree with ($k \in \mathbb{N} \wedge k > 2$), e.g., $k = 3$ in Peano [77]. The grid is created by recursively refining a cell into 2^{dim} children or by coarsening a family of 2^{dim} children into their parent cell.

Simple examples of a quadtree and an octree grid are shown in Fig. 2.2. To facilitate efficiently covering non-cubic domains, multiple trees can be combined into a *forest of octrees*. Libraries providing implementations of tree-based grids are, e.g., (i) Daino [78], (ii) Dendro [79], (iii) FLASH [80, 81], (iv) Gamer [82] (v) Octor [83], (vi) Parallel grid generator [84], (vii) `p4est` [85, 86], (viii) Peano [77], (ix) t8code [87], an extension to `p4est` for different element types such as triangles, tetrahedrons, and prisms using an extension to the Morton space-filling curve (SFC) [88], (x) TreElM [89] (part of the APES framework [90, 91]), (xi) Uintah [92], or (xii) waLBerla [53, 93].

Tree-based grids can be stored efficiently in a single bit per cell. This bit indicates for a depth-first search (DFS) traversal of the tree if the respective cell is refined. Additionally, traversing the grid in DFS order circumvents the problem of ambiguous cell-ordering.

[1] `http://math.boisestate.edu/~calhoun/www_personal/research/amr_software/`

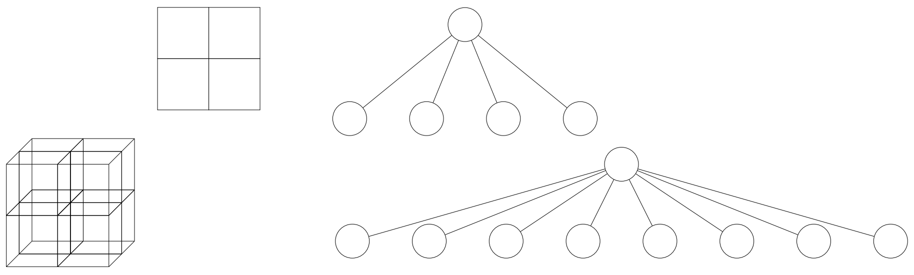


FIGURE 2.2 Simple example of a tree-structured Cartesian grid in 2D and 3D.

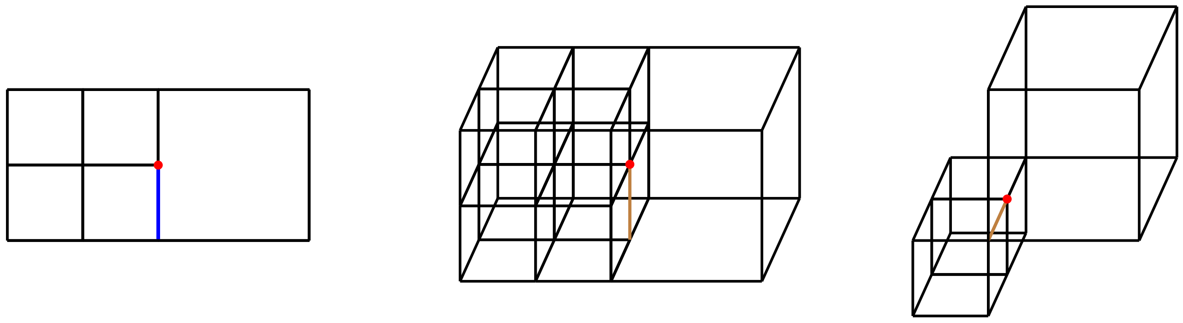


FIGURE 2.3 Illustration of all possible hanging entities in 2D and 3D. On the left, we show the 2D case. The blue face and the red corner hang across a face, thus we call them *face-hanging*. The same holds for the 3D case shown in the center, where we also name face (blue), edge (brown), and corner (red) *face-hanging*. On the right, we show hanging entities across an edge, which may only occur in 3D. Following the logic from before, we refer to the brown edge and the red corner as *edge-hanging*.

This issue would arise if cells of varying mesh-width were ordered lexicographically (as it is common for regular Cartesian grids). Usually, DFS order is associated with linearizing cells along a SFC. Different SFCs are used for cell ordering, e.g. (i) Morton-curve [94] (despite being discontinuous, at most two disjoint partitions are created [95]), (ii) Hilbert curve [96], (iii) Peano curve [97], or (iv) Sierpiński curve [98]. More detailed information about SFCs can be found in [99]. Note, that a bijective mapping between grid, tree, and bit-code exists for each curve. Individual bit-codes, however, vary between the respective curves for a given grid. Even if the Hilbert curve is the SFC with the best locality, performance is not significantly increased compared to the Morton curve [55, 100].

We refer to entities, i.e., faces, edges, or corners, at refinement boundaries that do not have a matching counter part on the other side of the refinement boundary as *hanging entities*. Hanging entities may occur at faces and edges, but not at corners. At faces, we refer to hanging faces, edges, and corners as *face-hanging entities*. Analogously, we refer to hanging edges and hanging corners at edges as *edge-hanging entities*. We illustrate all cases of hanging entities in 2D and 3D in Fig. 2.3.

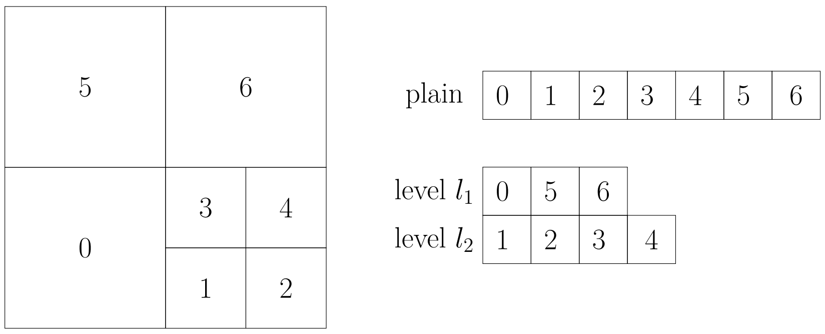


FIGURE 2.4 We illustrate two different ways for storing cell-centered data in a tree-structured Cartesian grid. We can store data in a linear list, “plain”, or we can separate the data by level.

2.1.2 Handling Data

To perform simulations, spatial and temporal discretizations have to be amended with a way to associate data with points in the grid. To make use of the locality offered by the SFC, storing data is closely related to the order of traversing the grid. Thus, data of payload and grid have to stored in the order of the SFC. Implementations differ in terms of the amount of tree-topology information that is stored. While some implementations only store leaf cells [79, 85], others store the full tree which might be partially replicated among different ranks [77, 78, 101]. In the latter case the local copy of the tree gets pruned to ensure minimal overhead for replication.

Evaluating stencil-based algorithms requires associating data with different entities in the grid ranging from cell volumes to cell corners. Additionally, data must be accessible from different cells to evaluate stencils. If data are associated with cell volumes, we can store them along with the metadata of the grid. This, however, impairs the locality of both, numerical payload and cell metadata. A simple solution is separating grid metadata from payload. We can store the payload linearly, following the SFC, or we can split the payload and follow the SFC separately for each level. We illustrate both options in Fig. 2.4.

Another common problem for stencil codes is accessing data of neighboring cells. All cells have a unique (level, index)-tuple within the tree, encoding their size and position. The most efficient option in terms of memory efficiency is calculating the respective index of neighboring cells and search for the index in the list of cells [86]. As neighbor relations only change after adapting the grid, avoiding the cost of searching and replacing them with $\mathcal{O}(1)$-lookups is straight-forward. To this end, we can either store pointers to neighboring cells [100] or use look-up tables to store cell-indices of neighboring cells [1]. Even more involved concepts where vertex data are loaded from a stream and shared between cells via stacks [77] exist. The idea here is that while traversing the grid along the SFC, one can algorithmically derive where to obtain the required data. Initially, the data will be read from an input stream. After the first cell has processed the data, they will be written to a specific stack. Here, they can be read from the next cell requiring access and be written to another specific stack. After the last cell requiring access has processed the data, they are written to an output stream. This output stream serves as an input stream in the next grid traversal. In this work, we use separate containers for payload and grid metadata. To find neighbors, we create look-up tables.

2.1.3 Important Algorithms

Evaluating stencil-codes requires five algorithms supported by the grid library. We want to (i) *create* and (ii) *partition* a grid among the ranks in parallel. Additionally, we want to (iii) restrict the level difference between neighboring cells to one *(2:1 balancing)* for two reasons. First, we want to avoid large local differences in grid resolution to avoid numerical artefacts. Second, we want to limit the number of cases in neighbor search for populating look-up tables. To evaluate stencil-codes independently for each local domain, we want to (iv) replicate at least one layer of the discretization and payload from neighboring processes adjacent to process boundaries *(ghost layer)*. Additionally, we require means to (v) *adapt* the grid in parallel during run time.

We use the naming scheme from [79] to describe the respective algorithms. They have been extended to forest of octrees in [85].

Locally creating the grid. Sundar et al. distinguish two different approaches for creating a tree-structured grid in parallel [79]. The *top-down* approach creates a grid by recursively splitting one or more root cells according to one or more refinement criteria. During the process, cells are re-distributed among the ranks. Note, that this problem may be mitigated by constructing a regular grid of level ℓ which results in $2^{\dim \ell}$ cells [83], where dim is the spatial dimension. Thus, cells of the regular grid may be used as a starting point for the top-down approach. By contrast, grids may also be created using a *bottom-up* approach. Here, points are converted into cells of the finest level, sorted according to the SFC, and distributed among the processors such that the load on all ranks is approximately the same. These cells serve as anchors for constructing the grid. The grid is constructed by filling the space between two consecutive cells with the coarsest possible tree. Note, that it suffices to choose one point per rank [79].

Traversing the grid. There are two natural ways for traversing a tree, depth-first and breadth-first. The former descends in the tree whenever possible while the latter explores the tree by level. The depth-first algorithms can be implemented in different ways. For binary trees, there are three fundamental ways for traversing the tree recursively using depth-first search. Each traversal consists of three steps, visiting the root and traversing both subtrees. We always traverse the left tree before the right tree. This leaves three possibilities when to visit the node:

```
1 function traverse_tree_dfs(tree)
2     traverse_tree_dfs(left_subtree)
3     traverse_tree_dfs(right_subtree)
4 end function
```

We can insert the call for visiting the current root before lines 2 (preorder, before visiting subtrees), 3 (inorder), or 4 (postorder) [102, 103]. Generalizing the idea of preorder and postorder traversals to trees with more than two subtrees is straight-forward by iteratively traversing all subtrees before or after visiting the node.

Parallel partitioning of the grid. The global ordering imposed by the SFC is commonly used to partition the grid among the ranks (*chain-on-chain* partitioning) [104]. To this end, the SFC is split into n_{proc} chunks of equal "length" according to a user-defined metric. Each process locally determines the weight of each cell as defined by the metric. The sum of local cell weights is accumulated by global reduction, e.g., `MPI_Allreduce` [46], and distributed among all ranks. The cells' weights are equally distributed among ranks by partial reduction, e.g., `MPI_Scan` or `MPI_Exscan` [46]. This way, we can accumulate cell weights in parallel on each rank and obtain updated chunks. These chunks determine where the cells are obtained from or migrated to.

2:1 balancing. If neighboring cells may have arbitrary differences in size, this will lead to two different effects. First, we have large local differences in the grid resolution. Second, finding neighbors is more involved, because a lot of different cases have to be distinguished. Both issues can be resolved by limiting the maximum difference in mesh-width for adjacent cells. In the context of this thesis, the maximum difference is chosen to be one. Sundar et al. propose an algorithm that first balances the local cells of a rank and then balances process boundaries. For local balancing, they present two approaches [79]: For each quadrant, we may search for any neighbors violating 2:1 balancing and refine those quadrants accordingly. The second approach is to virtually place the coarsest possible balanced cells as neighbors and remove overlaps. A hybrid implementation avoids costly search operations as well as large numbers of duplicates. Both approaches require several grid iterations.

To balance the grid across process-boundaries, the *prioritized ripple propagation* algorithm [105] can be used. It is important to note that there are no guarantees about the locality of the ripples. Additionally, balancing introduces iterative communication, as ripples may propagate over multiple ranks. To mitigate the issue, Sundar et al. propose a

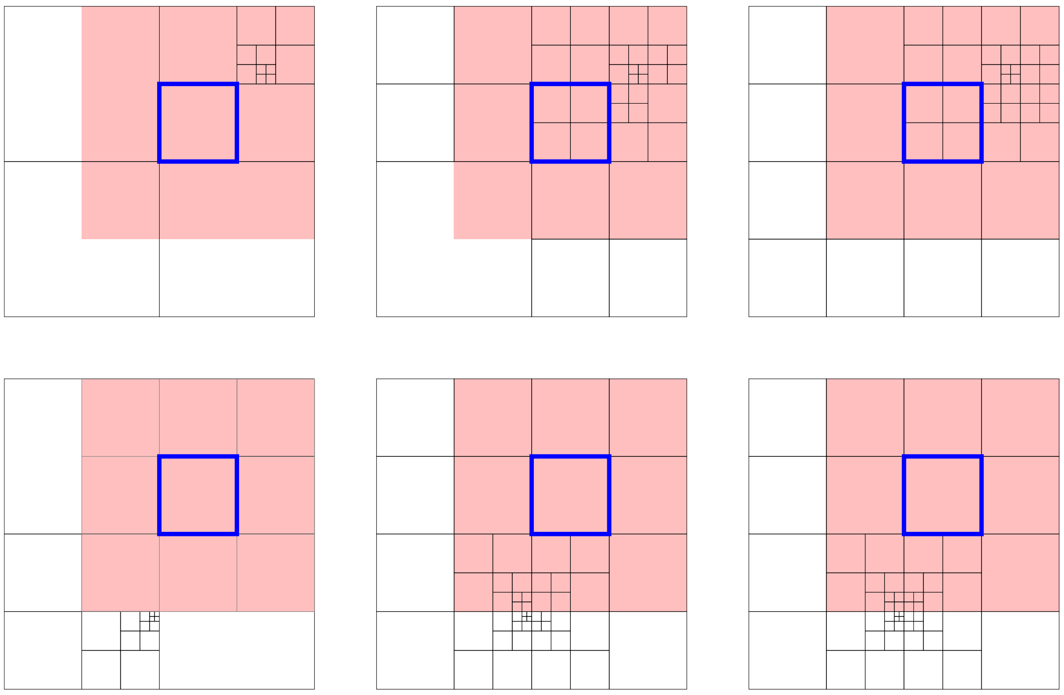


FIGURE 2.5 On the left, we show two exemplary grids as input for the balancing algorithm and the effects of refined cells inside (top) and outside (bottom) that cell's insulation layer (red area around highlighted cell). In the middle, we see balanced grids across faces and on the right balanced grids across faces and corners. In the upper row, the highlighted cell is split after balancing. The graphic was created with the help of an octree-visualization software for the `p4est` library [106].

two-step approach which introduces redundancy and communication overhead but avoids iterative communication. First, each rank communicates each local cell of a parallel boundary to all ranks overlapping with the cell's *insulation layer*. Second, each rank communicates all local cells overlapping with the insulation layer of a cell received in the first step to that cell's owner.

The insulation layer is the only region in the domain where there might be cells requiring the current cell to split [79]. It is the space spanned by the cell itself and virtual copies of that cell in all eight or 26 directions directly adjacent to it. Besides face neighbors, which have to be considered at all times, we are free to additionally include edge neighbors or edge and corner neighbors, depending on whether we want to enforce 2:1 balancing also across these entities. This, however, does not affect the size of the insulation layer. We consider two examples for balancing across faces and across faces and corners in Fig. 2.5. After exchanging cells, another local rebalancing step ensures that the grid is globally 2:1 balanced.

This approach is improved by Isaac et al. in [107] by minimizing the cost of repeatedly inserting newly created cells in the octree structure. They compress the octree and, thus, reduce the search space where a cell has to be inserted using *preclusion*. Cell a precludes cell b if and only if the parent of a is an ancestor of the parent of cell b or equal to b's parent cell. The advantage compared to the above algorithm is that it can continuously work on the smallest possible set sufficient to reproduce the final balanced octree.

Ghost layer and exchanging ghost data. When performing numerical simulations, the evaluation of the stencil inflicts data-dependencies between different partitions. To avoid infeasibly frequent communication, it is beneficial to replicate cells at the parallel boundary on neighboring processors. If we additionally store where which cell is replicated, data can be exchanged by point-to-point communication between processes at the end of a full iterative step. Codes using points for constructing their grid and store a truncated tree-topology on each rank, e.g., when implementing a fast multipole method (FMM) [108], may create ghost cells by locally replication. To this end, they send the coordinates of those points ending up in cells at the partition boundary to the respective neighboring processes. Then neighboring processes can locally construct the respective ghost cells [109].

An alternative way to construct a ghost layer for 2:1 balanced grids maps all cells to a virtual regular grid of the finest level. This creates a globally consistent naming scheme for all cells. We use this naming scheme to virtually create half-sized neighbors of all leaf cells adjacent to a process boundary in the tree. As partition boundaries are globally known and the tree is known to be balanced beforehand, neighboring cells on neighboring processes must not be finer than those virtually created. Thus, we can determine the owner of each virtual neighbor cell using binary search. Finally, we communicate the respective boundary cells to all neighboring processes that have been found [85].

This algorithm can be improved and generalized to arbitrarily refined grids [86] by recursively determining if a leaf touches the subdomain covered by a process. To this end, we traverse the tree using breadth-first search (BFS) (or a top-down approach) until all leafs have been reached. This allows reducing the search space by truncating all children that do not match the given search criteria. For creating the ghost layer, we abort the recursion for a cell if all its children end up on the same node. This is most effective, when evaluating multiple search criteria at the same time. When reaching a leaf-node, we check whether it is part of the parallel boundary and add the communication relation accordingly if needed [86, 87].

Dynamic grid adaptivity. To dynamically change the grid during simulation, we refine one cell into its 2^{dim} children or reduce a set of 2^{dim} cells to its parent cell and map, i.e., interpolate or restrict, the cells' payload accordingly. To this end, we evaluate one or more criteria for refinement and coarsening per cell or cell-group and store the result. We obtain three different outcomes (refine, keep, or coarsen) and, thus, have to define an order if different criteria yield different results.

If coarsening is executed before (e.g. [85]) or after (e.g. [55]) 2:1 balancing depends on the implementation. The latter avoids implementing a balancing algorithm for arbitrary grids, because coarsening may be denied based on a cell's neighbors. This approach requires accessing information about the size of neighboring cells during coarsening but avoids refining cells that have been coarsened immediately beforehand.

Mapping the numerical payload between both grids may happen either during actually refining or coarsening cells or in a separate grid traversal. In the second case, we have to keep track of the operation that was performed on the cell associated with the payload. As locally changing the grid generally introduces load imbalances, the grid is repartitioned after it has been adapted.

2.2 The `p4est` Library

In this work, we choose the grid framework `p4est`[2]. `p4est` is written in C and uses MPI for distributed memory parallelization. It is released as free software licensed under the GNU General Public License (GPL) in version 2.

`p4est` provides efficient and scalable implementations of the adaptive mesh refinement (AMR)-algorithms described above [62], contained in a well-designed library. Additionally, it is flexible through the forest-of-octrees approach. `p4est` stores cells in Morton-order. This makes `p4est` a good choice to integrate tree-structured grids into existing applications with ease, because Morton-order allows accessing neighbors comparably easy.

The library consists of four central elements: The macro-structure, i.e., the way octrees are arranged and connected to each other, is stored in `p4est_connectivity`. To uniquely identify elements, `p4est` uses a fixed naming scheme w.r.t. the local coordinate system of each tree which is shown in Fig. 2.6. This allows different octrees to have different local orientations by simply plugging them together. `p4est_connectivity` allows, e.g., to connect face f_1 of octree t_i to face f_3 of octree t_j. Based on the macro-structure, the micro-structure, i.e. the concrete discretization of the grid, is stored in the `p4est` data-structure. While the micro-structure can be arbitrarily adapted throughout the simulation, the macro-structure remains untouched by grid-adaptivity. The local orientation is fixed within each tree, i.e., all cells have the same local coordinate system.

Adding one or more layers of replicated cells on neighboring processes for efficiently performing simulations in parallel, is implemented in `p4est_ghost`. Here, a sparse matrix scheme, similar to compressed sparse row (CSR), is used to store which local

[2]`http://p4est.org/`

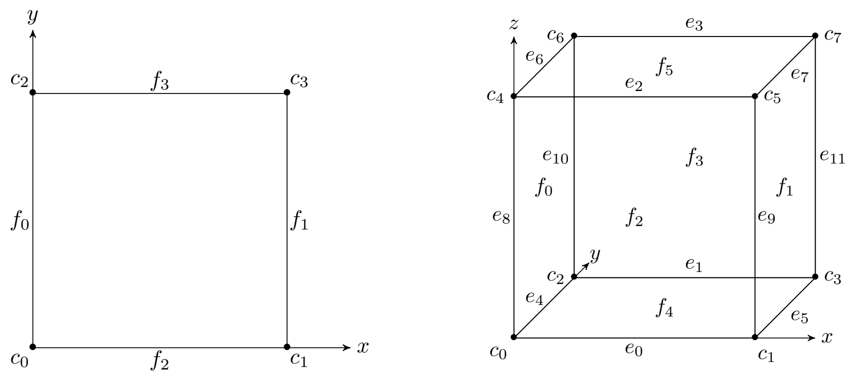
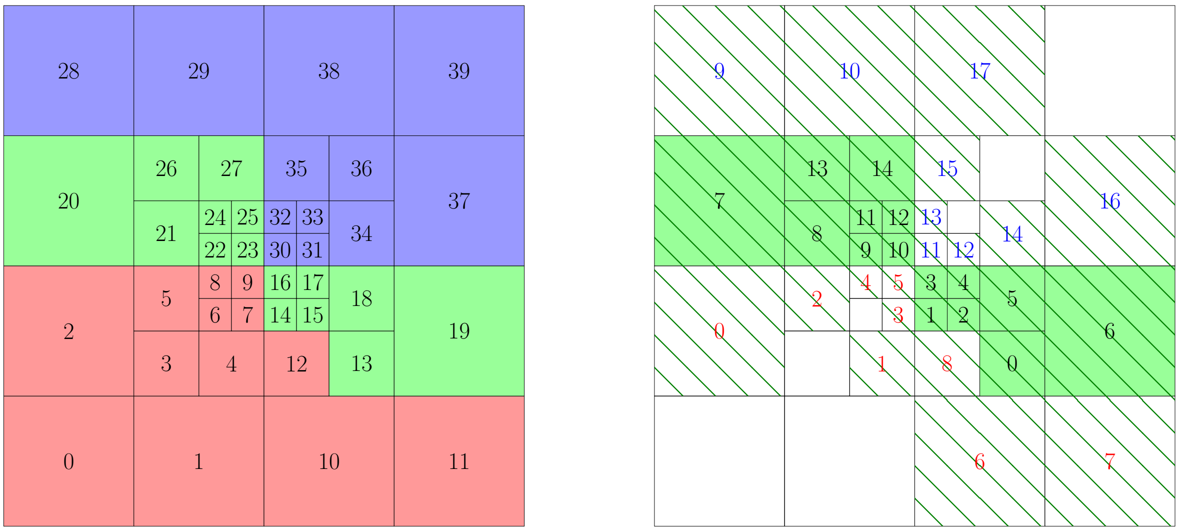


FIGURE 2.6 Naming scheme of faces, edges, and corners used by `p4est` in 2D and 3D.

cells have to be sent as replicates to other processes *(mirror cells)* and how many replicas are received from which process *(ghost cells)*. This scheme allows exchanging ghost-data by asynchronous point-to-point communication which is crucial for overlapping communication with computation *(communication hiding)*. We sketch the ideas of the sparse matrix scheme used in `p4est_ghost` in Fig. 2.7.

For traversing the grid, `p4est` provides an efficient iterator which can not only traverse the grid cell by cell but also visits each face, edge, and corner using recursion. The naive implementation for finding cell neighbors using the Morton curve would be to calculate the cell index of a neighboring cell and perform a binary search for a cell covering the respective area. `p4est` provides an improved algorithm in `p4est_iterate`. The algorithm traverses the forest top-down and enumerates all interfaces between cells exactly once [86]. This allows to efficiently identify adjacent cells when visiting interfaces between cells. `p4est_iterate` allows executing user-defined callback functions at each cell volume and at each interface.

`p4est` allows to dynamically change the grid by offering optionally recursive refinement and coarsening functions. Recursive refinement is implemented as a preorder depth-first traversal. This allows creating the grid shown in Fig. 2.8 in one grid traversal from the root A. Leafs in the tree are indexed by numbers, matching the SFC traversal order of the grid. Parent nodes, that `p4est` does not store in its leaf-only storage scheme, are indexed by capital letters using a depth-first traversal order. To create the grid in a single tree-traversal using recursive refinement, we start from root A, refining it. Then, we traverse A's children, beginning with child node B which we refine. We visit B's first child (node 0) and do not refine it and so forth. Analogously, we can recursively coarsen the given tree to node (A) in a single postorder depth-first traversal. Here, we visit each node after we visited its children and count the number of children that are leaf nodes Thus, we start with node A, descend to its first child node B and further to nodes 0, 1, 2, and C. In node C, we descend to nodes 3, 4, 5, and 6, identify them as leaf nodes and



ghosts	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
proc_offsets	0	9	9	18															
mirrors	0	1	2	3	4	5	6	7	8	9	10	12	13	14					
mirror_proc_mirrors	0	1	2	3	5	6	7	8	9	10	3	4	5	6	7	10	12	13	14
mirror_proc_offsets	0	10	10	19															

FIGURE 2.7 Schematic representation of the sparse matrix scheme used for storing communication patterns in `p4est_ghost`. We illustrate data-structures and locally existing cells for an exemplary grid of 40 cells distributed among three processes. On the left, we show the grid and a space-filling curve (SFC)-based domain decomposition based on the Morton ordering of cells. On the right, we show the local perspective of rank 1 (green). Its local cells are depicted in green while we hatch the share of the domain for which rank 1 knows the discretization. Local and ghost cells are locally ordered according to the global SFC-based ordering. Below, we show the data-structure that stems from this grid. Replicas of remote cells are collected in `ghosts`. Local cells, which will be replicated on other ranks, are copied to the `mirrors` array. The `proc_offsets` array is used to properly allocate receive-buffers. Its contains one more element than ranks in the system. At index i, we store the first element that is obtained from rank i. Thus, rank 1, e.g., obtains from rank 0 $(9-0) = 9$ cells or $(9-9) = 0$ cells from itself. The same idea holds for allocating send buffers. Here, `mirror_proc_offsets` indexes into `mirrors_proc_mirrors`. This vector then stores indices from the `mirrors` array which stores pointers to the local cells that have to be sent to neighboring processes.

return to node C. We coarsen nodes 3, 4, 5, and 6 to node $\tilde{C}$ and ascend to node B. Now, we coarsen nodes 0, 1, 2, and $\tilde{C}$ to node $\tilde{B}$. Following this traversal scheme allows to finally ascend to A and coarsen its four children into node $\tilde{A}$.

2.3 Summary

In this chapter, we have settled the background of the spatially-adaptive discretization that is used in this work. We have briefly reviewed different ways for realizing spatially-adaptive grids. We decided that dynamically-adaptive tree-structured Cartesian grids best suit our needs in dynamic simulation scenarios. Additionally, we have outlined an interface of important algorithms and presented different ideas for realizing them within a grid library. In the following, we choose to use the implementations provided by `p4est`, because `p4est` is not only well-structured and flexible but also elaborate, fast, and scalable. In the following chapters, we introduce the physical models that we port to adaptive grids, present ways to extend their discretizations to spatially adaptive grids, and give details on extending `p4est` to impelement the previously described extensions.

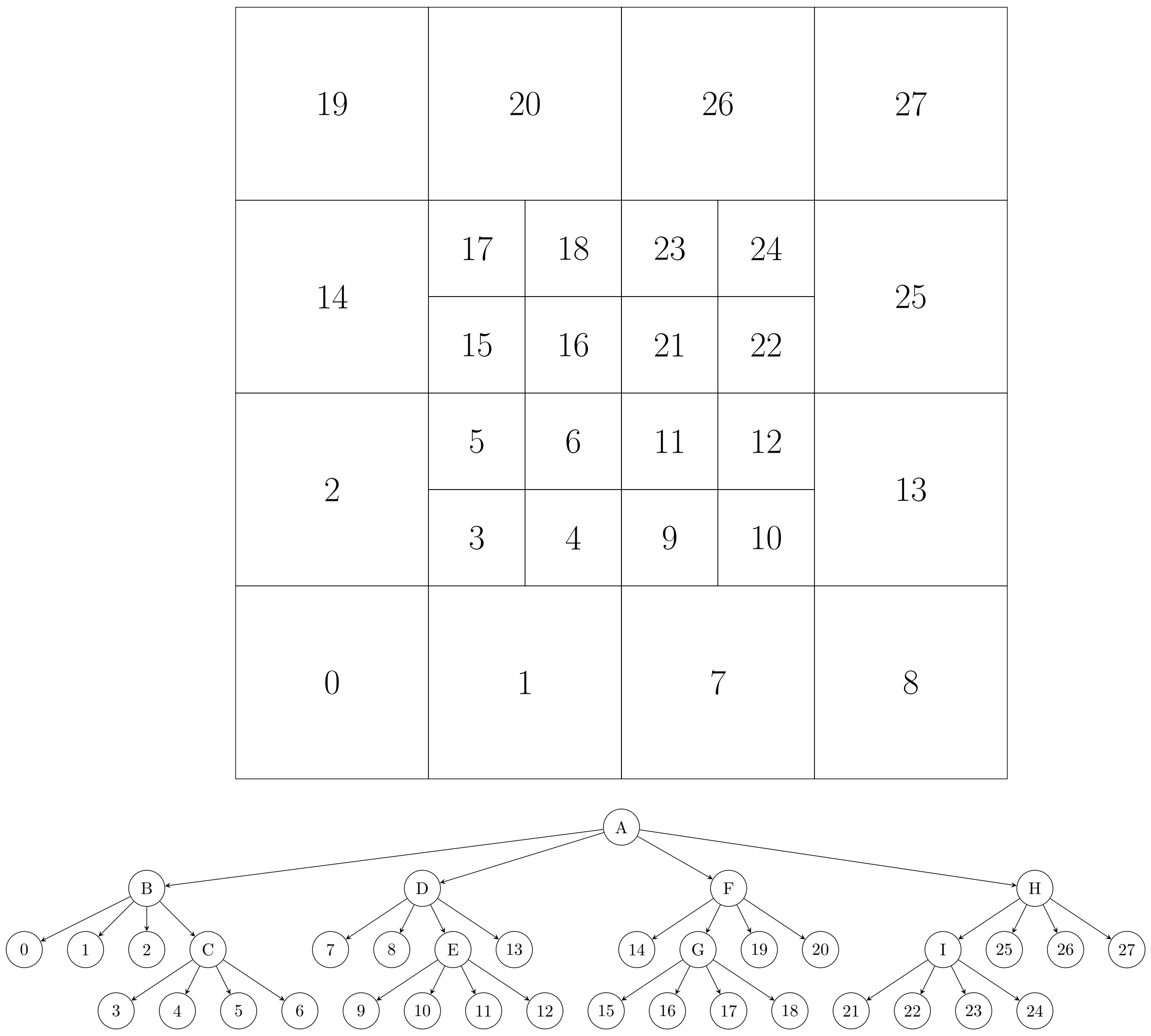


FIGURE 2.8 Exemplary grid with its corresponding tree to illustrate recursive coarsening and recursive refinement. Leafs in the tree are indexed by numbers, matching the SFC traversal order of the grid. Parent nodes that `p4est` does not store in its leaf-only storage scheme are indexed by capital letters in a depth-first traversal order. Recursive refinement allows to create this tree in a single iteration. Analogously, recursive coarsening allows to coarsen the given tree to the root node in a single grid traversal.

3 Target Software, Models, and Algorithms

In this chapter, we present several physical models which we use as building blocks to model our target application. Each model has a certain data access pattern and requires certain communication steps. To model the full system, we need an interaction layer on top of the individual models as well as an order in which the individual steps are executed within a time step. Different subsystems may even have different time step sizes. These factors result in a complex code-base, where different subsystems have to be suitably modeled and coupled to each other.

To observe multi-scale phenomena in the first place, we face the challenges of a large scenario, among other things, a huge amount of data, the coordination and load-balancing for several thousands of processes. In the following, we introduce the ESPResSo soft matter simulation software which we used to model our target application. We present the important aspects of the physical models and their coupling the way they are realized in ESPResSo.

3.1 Target Software ESPResSo Simulation Software

To simulate ionic liquids, potentially containing charged particles, we use ESPResSo[1], the **E**xtensible **S**imulation **P**ackage for **Res**earch on **So**ft Matter [31–33]. ESPResSo is released as free and open-source software under the GNU General Public License (GPL) in version 3. ESPResSo's physical algorithms are written in C++. Message Passing according to the MPI standard [46] enables parallel simulations. Users write Python scripts to set up and control their simulations. Core and simulation scripts exchange data via Cython [110] (C-extensions for Python). Some algorithms support using a single graphics processing unit (GPU) which is programmed based on NVIDIA's CUDA

[1]`http://espressomd.org/`

platform [47]. ESPResSo was originally developed as a molecular dynamics (MD) code but has been extended to a versatile and feature-rich simulation toolbox for a multitude of soft-matter scenarios over the years. ESPResSo has been used to simulate a large range of problems spanning from soot aggregation [111], cabin air filtration [112], hydrogels [113], biological membranes [114], DNA like-charge attraction [115], DNA translocation [30], to ionic liquids [116, 117].

3.2 Electrokinetic Equations

The bulk phase of the DNA translocation experiment described in Sec. 1.2 is modeled by the electrokinetic (EK) equations. The first continuous representation was introduced by Capuani et al. in [118]. The model consists of three main components: (i) Ionic flux, (ii) electrostatic potential, and (iii) hydrodynamics. For two ionic species, a positive ionic species denoted by index '+' and a negative species denoted by index '−', using the notation from [29], the EK equations read:

$$\partial_t c_\pm = -\nabla \cdot \boldsymbol{j}_\pm \tag{3.1}$$

$$\boldsymbol{j}_\pm = \underbrace{-D_\pm \nabla c_\pm - \mu_\pm z_\pm e c_\pm \nabla \Phi}_{j_\pm^{\text{diff}}} + \underbrace{c_\pm \boldsymbol{u}}_{j_\pm^{\text{adv}}} \tag{3.2}$$

$$\nabla \cdot (\varepsilon \nabla \Phi) = -\sum_\pm z_\pm e c_\pm \tag{3.3}$$

$$\eta \nabla^2 \boldsymbol{u} = \nabla p + \underbrace{\sum_\pm z_\pm e c_\pm \nabla \Phi}_{-f} \tag{3.4}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{3.5}$$

Equation (3.1) is the continuity equation for the species density fields $c_\pm$ and their ionic flux $\boldsymbol{j}_\pm$. Equation (3.2) is a diffusion-advection equation for the total ionic flux. The diffusive component consists of the species valencies $z_\pm$ and the electrostatic potential Φ, e denotes the elementary charge. The diffusion coefficient $D_\pm$ and the species mobility $\mu_\pm$ are related by the Einstein-Smoluchowski relation [119, 120] ($D_\pm/\mu_\pm = k_B T$, where k_B denotes the Boltzmann constant and T the electrolyte's absolute temperature, which is assumed to be constant). Advection refers to ions drifting along the fluid velocity field $\boldsymbol{u}$. Depending on the system's characteristic length scale one or the other component dominates. This relation, i.e., the relative share of diffusion compared to advection, is described by the Péclet number (advective transport rate over diffusive transport rate). For our system, where the characteristic length scale is in the nanometer range, the diffusive component dominates.

Equation (3.3) is a Poisson equation for the electrostatic potential Φ. The electric permittivity is defined as a product $\varepsilon = \varepsilon_0 \varepsilon_r(\boldsymbol{x})$, where ε_0 is the vacuum permittivity and $\varepsilon_r(\boldsymbol{x})$ the relative local permittivity of the medium. All magnetic effects are excluded. This is a valid assumption for aeqeous solutions, because the currents are typically small [29].

As the fluid is within Stokes regime, it is sufficient to prescribe the fluid's motion by Stokes' equation (3.4). Equation (3.5) is the continuity equation for an incompressible fluid.

The model is numerically instable, because different contributions to the ionic fluxes and the fluid flow cannot cancel exactly. Rempfer et al. illustrate for a discretization based on the finite element method (FEM) that no valid polynomial degree can be found for the ansatz functions given a system in thermodynamic equilibrium where all ionic fluxes and fluid flows vanish, i.e., $\boldsymbol{u} = 0$ and $\boldsymbol{j} = 0$ [121]. They mitigate the issue of spurious flow by adding a gradient field to the force density

$$\boldsymbol{f} = -\sum_{\pm}(z_{\pm} e c_{\pm} \nabla \Phi), \tag{3.6}$$

which is absorbed into the pressure gradient. In particular, they extend the force density to

$$\boldsymbol{f} = -\sum_{\pm}(k_B T \nabla c_{\pm} + z_{\pm} e c_{\pm} \nabla \Phi), \tag{3.7}$$

which lets the force density vanish in thermodynamic equilibrium. This follows from the definition of the equilibrium state,

$$\boldsymbol{j}_{\pm}/\mu = -k_B T \nabla c_{\pm} - z_{\pm} e c_{\pm} \nabla \Phi = 0 \tag{3.8}$$

$$\boldsymbol{u} = 0. \tag{3.9}$$

Thus, in contrast to the definition of the force density in (3.4), the pressure field does not have to compensate the non-vanishing force-term. Compared to Equation (3.4), the modified Stokes equation reads

$$\eta \boldsymbol{\nabla}^2 \boldsymbol{u} = \underbrace{\nabla p' + \sum_{\pm}(k_B T \nabla c_{\pm} + z_{\pm} e c_{\pm} \nabla \Phi)}_{= \nabla p}. \tag{3.10}$$

This reduces spurious flow by several orders of magnitude.

In the following, we describe the individual components, their implementation in ESPResSo, and their coupling in more detail. We introduce the lattice-Boltzmann method (LBM) to simulate hydrodynamics in Sec. 3.3, the finite volume method (FVM) representation of the ionic flux in Sec. 3.4, and conclude with describing the discretization of the electrostatic potential using iterative methods such as successive over-relaxation (SOR) as well as discrete Fourier transform (DFT). These algorithms can be coupled to ESPResSo's molecular dynamics (MD) simulation engine. We introduce molecular dynamics in Sec. 3.6 and describe the coupling of the components in Sec. 3.8.

3.3 Hydrodynamics: Lattice-Boltzmann Method

3.3.1 Introduction

To solve the Navier-Stokes equations, ESPResSo ships an implementation of the lattice-Boltzmann method (LBM). Here, we will only give a short introduction on the central concepts. For a more detailed introduction, we refer to [17, 18]. Instead of directly discretizing the Navier-Stokes equations, the LBM is based on Boltzmann's equation (3.11).

$$\frac{df(\boldsymbol{x},\boldsymbol{v},t)}{dt} = \partial_t f + \nabla_x f + \frac{F}{m}\cdot\nabla_v f = \Omega(f) \tag{3.11}$$

The Boltzmann equation neither models the fluid based on macroscopic quantities nor microscopically by considering individual particles. Instead, the fluid is described mesoscopically by probability distributions f and a potentially arbitrary complex collision operator Ω. The probability distributions describe the probability of finding a virtual mesoscopic particle at a given point in time and space with a given velocity. Thus, the Boltzmann-equation spans a seven-dimensional space with independent variables position $\boldsymbol{x}$, moment $\boldsymbol{v}$, and time t.

The LBM offers a way to discretize Boltzmann's equation (3.11). To discretize space, LBM uses a regular Cartesian grid. The velocity space is discretized by restricting the velocity space to few distinct velocities linking grid nodes with each other. Typically, one of the four stencils shown in Fig. 3.1 is used and characterized according to the *DnQm* notation [122]. Here, n describes the spatial dimension and m the number of distinct velocities. With this, the LBM-equation can be formulated as follows:

$$\underbrace{f_i(\boldsymbol{x}+\boldsymbol{c}_i dt, t+dt) = f_i(\boldsymbol{x},t)}_{\text{propagation/streaming}} + \underbrace{\sum_k L_{ik}\{f_k^{\text{eq}} - f_k(\boldsymbol{x},t)\} + \Xi_i}_{\text{collision/relaxation}}. \tag{3.12}$$

Here, f_i denotes the densities of virtual particle populations of velocity $\boldsymbol{c}_i$. The density change of population f_i due to external forces is given by Ξ_i. We derive the local equilibrium distribution f_i^{eq} by a second order expansion of the Maxwell-Boltzmann equilibrium distribution

$$f_i^{eq} = a_i \rho \left(1 + \frac{\boldsymbol{u} \cdot \boldsymbol{c}_i}{c_s^2} + \frac{(\boldsymbol{u} \cdot \boldsymbol{c}_i)^2}{2c_s^4} - \frac{u^2}{2c_s^2} \right), \tag{3.13}$$

where a_i are prefactors, $\boldsymbol{u}$ denotes the local fluid velocity, and c_s is the speed of sound on the lattice. The choice of mesh width and stencil implicitly yields the time step, which is chosen such that a mesoscopic particle moves from one lattice node to the next in one time step. It can be proven by Chapman-Enskog expansion [123] that LBM is equivalent to directly solving the Navier-Stokes equations. We recover macroscopic fluid moments (density ρ, momentum $\boldsymbol{p}$, stress $\overline{\pi}$) from population densities by integrating over the probability densities

$$\rho = \sum_i f_i, \qquad \boldsymbol{p} = \sum_i f_i \boldsymbol{c}_i \qquad \overline{\pi} = \sum_i f_i \boldsymbol{c}_i \otimes \boldsymbol{c}_i, \tag{3.14}$$

where $\otimes$ denotes the tensor product. The fluid velocity $\boldsymbol{u}$ can be derived from the fluid's momentum $\boldsymbol{p} = \rho \cdot \boldsymbol{u}$. As the LBM is an explicit and matrix-free method without any global synchronization steps and a small stencil, it is excellently suited for parallelization.

The LBM is classically implemented as a two-step algorithm of *collision* and *streaming*. Depending on which step is executed first, the scheme is called *push* (collision precedes streaming) or *pull* scheme.

Collision. The collision step is a local interaction of populations which drives the fluid towards its local equilibrium. There are different schemes to model the collision operator L_{ik}. The simplest was introduced by Bhatnagar et al. where $L_{ik} = \lambda \delta_{ik}$. The so-called BGK (after the inventors Bhatnagar, Gross, and Krook) or single-relaxation-time (SRT) collision operator relaxes all populations with a single relaxation parameter λ which we choose according to the desired shear viscosity of the fluid. Different fluid-parameters such as the bulk viscosity cannot be chosen independently. This mitigation can be resolved by choosing more involved collision operators such as two-relaxation-time (TRT) [125, 126] or multi-relaxation-time (MRT) [127] schemes. Here, different linear combinations m_i of the LBM populations f_i *(modes)* are relaxed towards their local equilibria at different rates λ_i. Modes correspond to different macroscopic quantities such as fluid density, velocity, or stress. MRT is an actual superset of the simpler collision models TRT and SRT. Thus, MRT can recover those schemes when parametrized accordingly and is proven to be more stable than the simpler schemes [128].

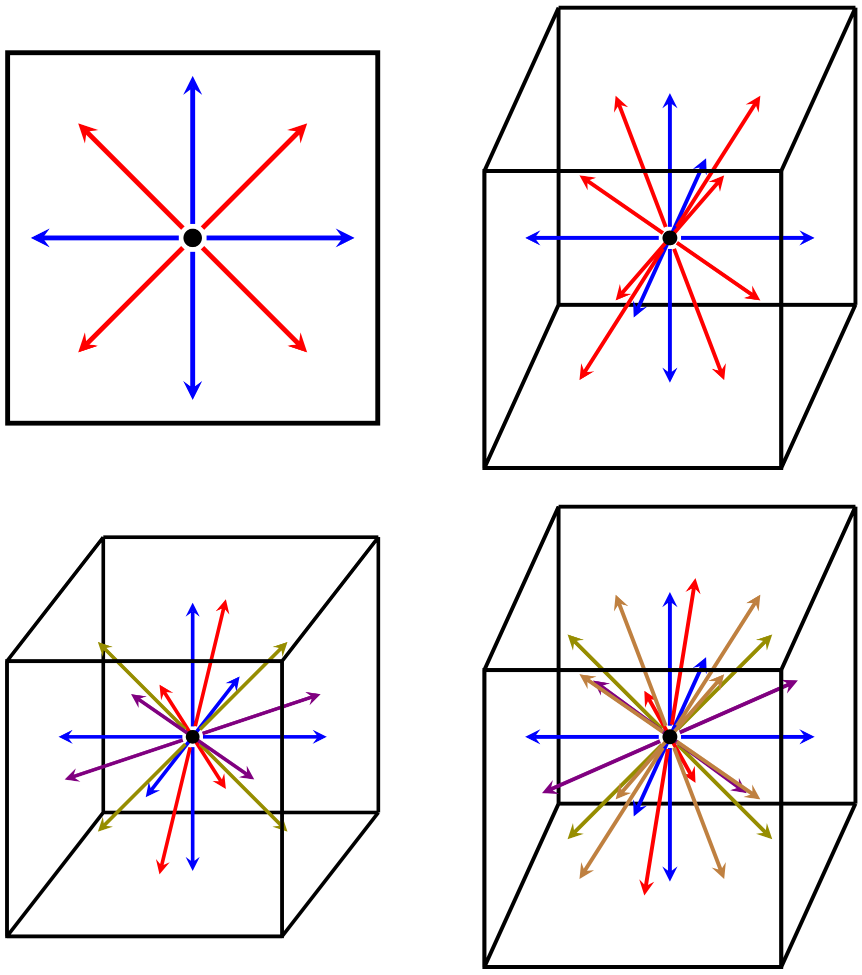


FIGURE 3.1 Popular LBM stencils: D2Q9 (top left), D3Q15 (top right), D3Q19 (bottom left), and D3Q27 (bottom right).

Streaming. During the streaming step, the results of the previous collision step are propagated to the next lattice sites *(acoustic scaling)*. As shown in Fig. 3.2 for a D2Q9 stencil, populations are directly transported to their neighboring cells.

Boundary conditions. Until now, we only considered the fluid domain, i.e., our system neither contains obstacles nor inflow or outflow regions. The simplest way of modeling obstacles are bounce-back boundaries. The idea of bounce-back boundaries is illustrated in Fig. 3.3: Populations entering the boundary are reflected back into the originating fluid node. This antisymmetry results in an effective fluid velocity of 0 at the boundary *(no-slip)*. More advanced boundary conditions can, e.g., be found in [129–132].

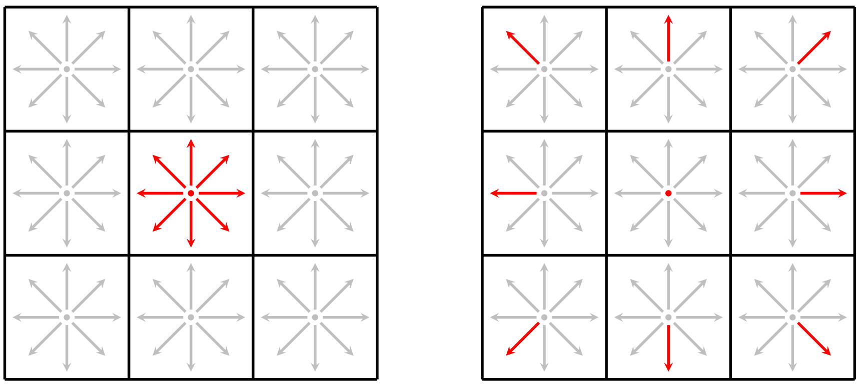


FIGURE 3.2 Streaming operation on a regular LBM grid using a D2Q9 stencil. Populations are directly transported to the cell in the direction of the respective discrete velocity.

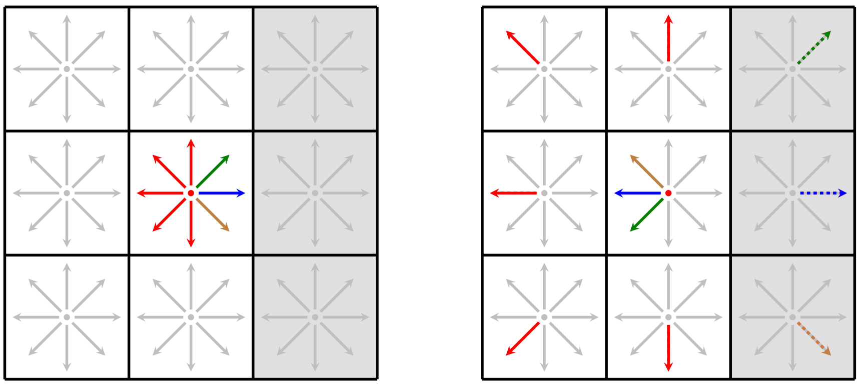


FIGURE 3.3 Bounce-back boundary conditions for a wall-boundary (gray) on a regular LBM grid using a D2Q9 stencil before streaming and bounce-back (left) and afterwards (right). The populations entering the boundary (dashed arrows on the right in brown, blue, and green) are highlighted. The highlighted populations are reflected back into the originating cell (solid arrows, colored accordingly).

3.3.2 Reducing the Limitations of the LBM

The classical LBM has several short-comings in terms of stability such as the low Mach number limit or the restriction to weakly compressible flows. Some of these issues can be mitigated, but going into detail about the concrete measures is beyond the scope of this work. Apart from improved boundary conditions, there are several ways to make the LBM numerically more robust. One option is to systematically derive a higher order equilibrium distribution [133]. Alternatively, we can use more stable LBM kernels such as cascaded LBM [134, 135] or, even more stable, cumulated LBM [136, 137] which resolve issues with Galilean invariance. A nice overview is given in [138].

Classically, the LBM suffers from a large memory footprint. This stems from using double-buffering schemes which avoid overwriting population densities f_i during streaming. Streaming in a particular order resolves these data-dependencies. By using

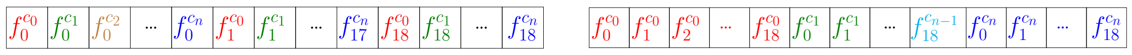

f_0^{c0} f_0^{c1} f_0^{c2} ... f_0^{cn} f_1^{c0} f_1^{c1} ... f_{17}^{cn} f_{18}^{c0} f_{18}^{c1} ... f_{18}^{cn}
f_0^{c0} f_1^{c0} f_2^{c0} ... f_{18}^{c0} f_0^{c1} f_1^{c1} ... $f_{18}^{c_{n-1}}$ f_0^{cn} f_1^{cn} ... f_{18}^{cn}

FIGURE 3.4 Struct-of-arrays-based storage scheme (streaming-optimized, left) compared to an array-of-structs-based storage scheme (collision-optimized, right) for a D3Q19 implementation of the LBM as it is used in ESPResSo. Colors encode the cell which the respective value is associated with.

single-buffering schemes [139, 140] the memory footprint of the LBM can be halved.

3.3.3 LBM in ESPResSo

ESPResSo ships an MRT-based implementation of the LBM using a D3Q19 stencil. Originally, the LBM was developed to subject a molecular ensemble to a background flow. To account for Brownian motion, the LBM can be thermalized by different probability distributions to generate noise [129, 141]. Data are stored in a streaming-optimized storage scheme [142]. Streaming-optimized refers to a struct-of-arrays data-structure in contrast to the collision-optimized storage scheme which is an array-of-structs data-structure. Figure 3.4 illustrates the difference between both schemes.

Obstacles are modeled by bounce-back boundary conditions. Those obstacles can have different shapes and the fluid cells overlapping an obstacle are marked as boundary cells. Inflow and outflow boundary conditions are realized in the same way. They are implemented similar to velocity Dirichlet boundary conditions and have an assigned velocity value. Instead of just mirroring the populations that have left the fluid, those populations are shifted according to the respective velocity difference during bounce-back. ESPResSo ships no implementations of further boundary conditions as, e.g., free outflow.

ESPResSo contains two different implementations, a central processing unit (CPU)-based implementation parallelized by a domain-decomposition approach using MPI. Additionally, ESPResSo contains a CUDA-based version for a single GPU [143].

3.4 Ionic Flux

The ionic flux in the continuous model by Capuani et al. consists of a continuity equation (3.1) and a diffusion-advection equation (3.2) based on the Nernst-Planck equation [118]. In ESPResSo, the diffusive and the advective component of the flux are discretized separately.

3.4.1 Diffusive Flux

Local differences in the ion density drive the diffusive flux. As it is crucial to conserve ionic densities to ensure constant net charge, ESPResSo uses a finite volume method (FVM) to discretize the diffusive flux [5, 29]. We calculate the flux to the neighboring lattice site in direction $+\boldsymbol{d}_i$ using a symmetric finite difference equation:

$$j^{\mathrm{diff}}_{\pm,i}(\boldsymbol{x}) = -D_\pm \frac{c_\pm(\boldsymbol{x}) - c_\pm(\boldsymbol{x}+\boldsymbol{d}_i)}{\|\boldsymbol{d}_i\|_2} - \mu_\pm z_\pm e \frac{c_\pm(\boldsymbol{x}+\boldsymbol{d}_i)+c_\pm(\boldsymbol{x})}{2} \cdot \frac{\Phi(\boldsymbol{x}-\boldsymbol{d}_i)-\Phi(\boldsymbol{x})}{\|\boldsymbol{d}_i\|_2}. \tag{3.15}$$

A D3Q18 stencil ensures consistency with the D3Q19 stencil chosen in the lattice-Boltzmann method (LBM) in terms of data-access patterns. Both stencils access data from neighbors across faces and edges, however, the D3Q18 stencil lacks a link with the cell itself. FVM schemes are known to fulfill conservation laws particularly well as they describe quantities in an integral formulation over cell volumes rather than pointwise approximations [24].

3.4.2 Advective Flux

The advective flux component describes the flux of ions which are dragged along the direction of the local flow field. ESPResSo uses a volume-of-fluid scheme to model advection. The scheme virtually displaces the fluid contained in each grid cell by the distance $\boldsymbol{s}$ which the fluid travels given its local velocity $\boldsymbol{u}$ in one time step Δt,

$$\boldsymbol{s} = \boldsymbol{u}\Delta t. \tag{3.16}$$

Assuming homogeneous ionic concentrations within each cell, a FVM scheme propagates ionic concentrations between neighboring cells. We calculate the relative size of the overlap between neighboring cell and virtually displaced cell. Then, we propagate the relative amount of ionic concentration to the neighboring cell, corresponding to the relative size of the overlap. We illustrate this algorithm in Fig. 3.5. The inherently diffusive behavior of this scheme is not an issue, because in our regime diffusion dominates advection.

3.4.3 Total Ionic Flux

The total ionic flux is calculated as a weighted sum of diffusive and advective flux. It is used to propagate ionic concentrations in a finite volume representation of the ionic continuity equation (3.1) between grid nodes

$$c_\pm(\boldsymbol{x}, t+\Delta t) = c_\pm(\boldsymbol{x}, t) - \Delta t \sum_i A_i \boldsymbol{j}_{\pm,i}(\boldsymbol{x}, t) \tag{3.17}$$

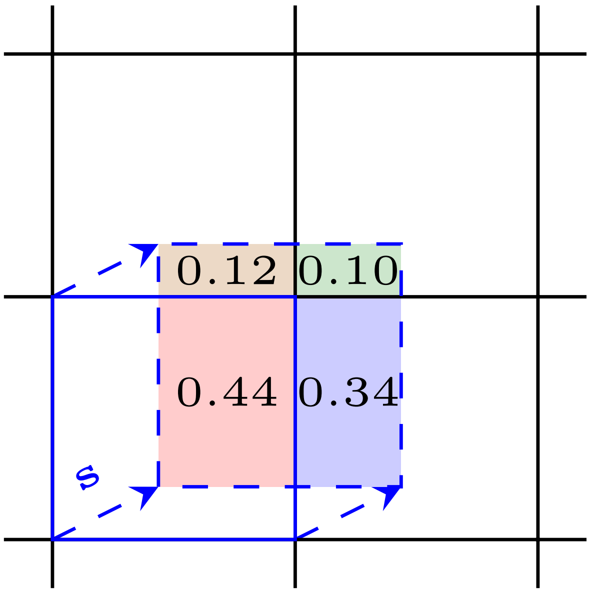


FIGURE 3.5 Illustration of the volume-of-fluid method used to calculate the advective flux. We virtually displace the highlighted cell by the distance $\boldsymbol{s}$ the fluid travels given its local velocity $\boldsymbol{u}$ within the time step Δt. We assume a homogeneous distribution of ions within each cell. Thus, we calculate the relative size of the overlap between the neighboring cell and the displaced image of the current cell. Then, we transfer exactly this amount of ionic concentration to that neighbor.

with

$$\boldsymbol{j}_{\pm,i}(\boldsymbol{x},t) = \boldsymbol{j}^{\text{diff}}_{\pm,i}(\boldsymbol{x},t) + \boldsymbol{j}^{\text{adv}}_{\pm,i}(\boldsymbol{x},t). \tag{3.18}$$

The weighting factors A_i are chosen such that the mean square displacement (MSD) of an ionic species is correctly reproduced for a fluid-at-rest and a neutral electric field. This scheme ensures conservation of ionic densities c down to arithmetic precision, because the inflow of one cell exactly equals the outflow of the neighboring cell [29].

3.5 Electrostatic Potential

We derive the electrostatic potential Φ from the ionic densities c using Poisson's equation:

$$\nabla \cdot (\varepsilon \nabla \Phi) = -\sum_{\pm} z_{\pm} e c_{\pm}. \tag{3.3}$$

Assuming constant electric permittivity ε, Poisson's equation (3.3) can be written as

$$\nabla^2 \Phi = -\frac{4\pi \ell_B k_B T}{e^2} \varrho \tag{3.19}$$

with net charge density ϱ and Bjerrum length ℓ_B where $\varrho = \sum_{\pm} z_{\pm} e c_{\pm}$ and $\ell_B = \frac{e^2}{4\pi\varepsilon k_B T}$. We discretize the Laplacian by a seven-point finite difference stencil on the same grid as the finite volume method (FVM) described above to avoid interpolation during coupling. This yields the following stencil

$$
\begin{aligned}
-\frac{4\pi h^2 \ell_B k_B T}{e^2}\varrho(x_0,x_1,x_2) = {} & \Phi(x_0+h,x_1,x_2)+\Phi(x_0-h,x_1,x_2) \\
& + \Phi(x_0,x_1+h,x_2)+\Phi(x_0,x_1-h,x_2) \\
& + \Phi(x_0,x_1,x_2+h)+\Phi(x_0,x_1,x_2-h) \\
& - 6\Phi(x_0,x_1,x_2).
\end{aligned}
\tag{3.20}
$$

with global mesh width h. To solve this scheme, several approaches have been compared in [144]. Here, methods based on discrete Fourier transform (DFT) were deemed best despite enforcing globally constant electric permittivity ε and, thus, globally constant Bjerrum length ℓ_B. As evaluating the Laplacian is turned into a set of fused multiply-add (FMA) operations in Fourier space, DFT methods yield accurate results at low computational cost compared to, e.g., iterative methods such as successive over-relaxation (SOR) which Capuani et al. use in their original implementation. To transform between Fourier space and back we use fast Fourier transform (FFT) [29].

3.6 Molecular Dynamics

All subsystems for modelling the electrokinetic (EK) equations in a continuous representation have been discretized on a grid. In the following, we will add discrete particles that are dissolved in the ionic fluid. In our target-application, this might be the DNA string.

3.6.1 Short-Range Molecular Dynamics

ESPResSo was originally designed as a molecular dynamics (MD) simulation software. Detailed introductions about MD simulations can, e.g., be found in [145, 146]. In MD, we trace the trajectories of particles that interact through different kinds of force fields. The movement of particles is governed by Newton's equation of motion. Theoretically, we would have to evaluate all forces between any arbitrary number of particles and the current particle p_k. These forces are derived from intermolecular potentials V for particle positions $\boldsymbol{x}_{p_i}$,

$$
f(p_k) = -\nabla_{x_{p_k}} V(\boldsymbol{x}_{p_0},\dots,\boldsymbol{x}_{p_n}).
\tag{3.21}
$$

In practice, however, we approximate the generic potential V by a pair potential U. Again, the force acting on an individual particle p_k is calculated as a sum over all attractive and repulsive forces between the current particle and all remaining particles in the system

$$f(p_k) = -\nabla_{x_k} V(\boldsymbol{x}_0, \dots, \boldsymbol{x}_n) \approx \sum_{i, i \neq k} -\nabla_{x_k} U(r_{ik}) = \sum_{i, i \neq k} \boldsymbol{f}_{ik}. \tag{3.22}$$

We designate the Euclidean distance between particles i and k by r_{ik}. Evaluating the potential yields a force f_{ik} between both particles. The sum over all intermolecular forces yields the force acting on particle p_k [147].

The computational complexity of this algorithm is $\mathcal{O}(n^2)$ where n is the number of particles in the system. For short-ranged potentials, this complexity can be reduced to $\mathcal{O}(n)$. An arbitrary pair-potential U is short-ranged if and only if

$$\int_{\|\boldsymbol{x}\|_2 > r_c} U(\|\boldsymbol{x}\|_2) d\boldsymbol{x} \xrightarrow{r_c \to \infty} 0, \tag{3.23}$$

i.e., if the influence of the potential can be neglected beyond a particle distance r, if r is large enough. Then, we can define a maximum distance r_c between two particles *(cut-off radius)*. For particle pairs which are farther apart than this distance, i.e., $r > r_c$, we neglect the force contribution of the respective interaction. Given such a potential, we can employ the *Linked-Cell method* [148, 149]. Here, the domain is discretized by a Cartesian grid with a mesh width of r_c. It can be further optimized using Newton's Third Law. By using an appropriate search pattern, we can guarantee to visit each particle pair exactly once and add the respective force to both particles. We refer to all 26 direct neighbor cells as *full shell neighborhood* and the optimized pattern based on Newton's Third Law with 13 neighbor cells as *half shell neighborhood*. If an exact matching between multiples of r_c and the spatial domain size is impossible, it is easier to choose a slightly larger mesh width than a smaller one. This guarantees finding all interacting particles within directly adjacent neighbor cells. We will, however, have to probe more particles outside the cut-off radius r_c. The fundamental idea of the Linked-Cell method as well as half shell and full shell neighborhood is illustrated on the right of Fig. 3.6.

Another way for reducing computational cost uses *Verlet lists* [150]. Here, we store for each particle a list of all particles within the sum of the cut-off radius and an additional distance *(skin)*. This skin allows reusing Verlet lists over multiple time steps.

A well-known and widely used short-range interaction potential is the Lennard-Jones

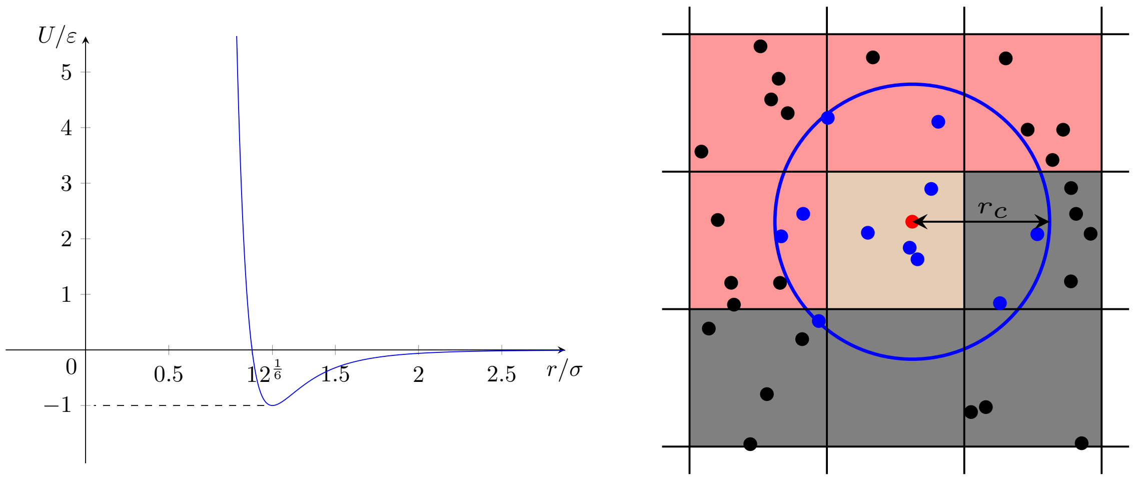


FIGURE 3.6 We print the LJ potential on the left where we plot the potential U over the particle distance r. ε steers the minimum of the potential and, thus, the strength of the bonds, while σ controls the distance where the potential's influence can be neglected.
On the right, we illustrate the basic idea of the Linked-Cell method. After calculating the distances of all particles (black and blue), in the current cell and in neighboring cells, to the red particle, only the blue particles (those within the blue circle) contribute to the force acting on the red particle. The half shell neighborhood is marked e.g. by the red or gray area, their union yields the full shell neighborhood. The pairs in the cell containing the current particle (brown) must be considered.

(LJ) potential [151, 152] which represents a special case of the Mie potential [153].

$$U_{LJ}(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right). \tag{3.24}$$

It represents two physical aspects, that is Pauli repulsion, modeled by $\left(\frac{\sigma}{r}\right)^{12}$, and van der Waals attraction, modeled by $\left(\frac{\sigma}{r}\right)^{6}$. The potential curve of the LJ potential is depicted on the left in Fig. 3.6. The parameter ε controls the depth of the minimum in the potential curve and, thus, the strength of the repulsive and attractive forces. This allows simulating materials with different properties. Larger values of ε lead to stronger bonds, i.e., harder materials. σ models the distance where the potential almost vanishes [147].

3.6.2 Long-Range Molecular Dynamics

Long-range interactions mostly refer to charged systems where we evaluate the Coulomb potential between particles. To reduce the computational cost, we can use Ewald summation [154, 155]. Further improvements can be made by interpolating the charges to a grid, e.g., the particle-particle-particle-mesh (P^3M) algorithm [148]. As we did not perform any simulations with charged particles over the course of this work, we will not go beyond mentioning that ESPResSo is capable of performing such simulation for the sake of completeness.

3.7 Parallelization

To perform large-scale simulations, we have to parallelize the aforementioned concepts. According to the message-passing paradigm, the grid-based concepts described in Sec. 3.3 to Sec. 3.5 use a domain decomposition approach. To this end, we slice the rectangular simulation domain into n_{proc} (number of processes in the system) smaller rectangles of fixed size. Then, we assign one such slice to each process. To make each domain independent of its direct neighbors, each domain is surrounded by a ghost layer, where information from neighboring processes gets replicated using MPI-based message-exchange.

Parallelizing molecular dynamics (MD) simulations offers more options. Similar to grid-based algorithms, we can employ a domain decomposition. However, we can also distribute the computational load based on the number of particles (atom decomposition) or on the number of force pairs (force decomposition). Domain decomposition is known to be the most efficient approach [36].

To this end, ESPResSo also has a domain decomposition method implemented which is fixed over the course of the simulation. To reduce the computational complexity of the short-range MD algorithm, ESPResSo uses a combination of a Linked-Cell grid with Verlet lists. Here, we set mesh width to the sum of cut-off radius and skin. Thus, the Linked-Cell method can be recovered by setting the skin-size to 0.

3.8 Interaction of Components

Up to now, we have described the individual subsystems. To simulate meaningful physical systems, we have to connect these subsystems by a coupling scheme and define an execution order within a time step. As described in the previous section, all components use the domain decomposition approach for parallelization. Thus, coupling can be performed locally on each process.

If we simulate a system that consists of all subsystems, ESPResSo executes the algorithms of the respective subsystems in the following order within a time step:

1. Short-range molecular dynamics (MD),
2. diffusive ionic flux,
3. advective ionic flux,
4. electrostatic potential and long-range MD, and

5. hydrodynamics.

Coupling between grid-based algorithms is given by the system equations (3.1) to (3.5). As we use the same grid for all three grid-based components of the electrokinetic (EK) model (ionic flux, electrostatic potential, and hydrodynamics), we can simply read the required values of the respective cells. For coupling short-range MD to the grid algorithms, ESPResSo implements the scheme by Dünweg et al. [156].

An electric field $\boldsymbol{E}$ applies a potential force based on the net charge density ϱ on the fluid given as

$$\boldsymbol{f}_{\text{pot,fl}} = \varrho \boldsymbol{E} = -\sum_{\pm} z_{\pm} e c_{\pm} \nabla \Phi. \tag{3.25}$$

We derive a coupling-force from the local velocity difference of particle and fluid as a bi-directional frictional force coupling

$$\boldsymbol{f}_{\text{fl},p_i} = -\Gamma(\boldsymbol{v}_i - \boldsymbol{u}(\boldsymbol{x}_j, t)) + \boldsymbol{f}_{\text{st},i}. \tag{3.26}$$

The force $\boldsymbol{f}_{\text{fl},p_i}$ between particle p_i and fluid is given by the friction coefficient Γ times the velocity difference between particle and fluid. We determine the local fluid velocity by trilinear interpolation [157]. To model Brownian motion, we add a stochastic force $\boldsymbol{f}_{\text{st},i}$. The sum of both forces and a global external force forms the density change due to external forces Ξ_i in equation (3.12)

$$\Xi_i = \boldsymbol{f}_{\text{global}} + \boldsymbol{f}_{\text{pot,fl}} + \boldsymbol{f}_{\text{fl},p_i}. \tag{3.27}$$

3.9 Summary

In this section, we have described the physical models describing our target application. We have introduced the ESPResSo simulation software that can model this system. Additionally, we have described the status quo of this software: Both in terms of how ESPResSo discretizes and implements the respective models as well as how ESPResSo realizes the coupling of the individual subsystems. In Chap. 4, we generalize the grid-based algorithms to adaptive grids. This defines constraints that have to be met by the `p4est` library. We describe our extensions to `p4est` in order to meet these constraints as well as our integration of `p4est` with ESPResSo in Chap. 5.

4 Adaptive Discretization of Physical Models

In this chapter, we generalize the algorithms and discretization schemes presented in the previous chapter to adaptive grids. We present adaptive formulations for hydrodynamics, ionic flux, and the evaluation of the electrostatic potential. Additionally, we explain how an adaptive representation of the respective models affects coupling between subsystems. Each extension defines constraints on the underlying adaptive discretization.

4.1 Hydrodynamics

The fundamental idea of the adaptive lattice-Boltzmann method (LBM) schemes that we present in the following, is to consider different discretization levels as separate, locally regular grids. The union of the respective grids yields the spatial domain. To transmit information between these different grids, we introduce an overlap region at refinement boundaries. This region allows transferring information between different grid levels.

4.1.1 Interpolation Schemes

Interpolation-based LBM schemes [158–160] locally embed regions with finer grid resolution into a global coarse grid. The smaller grid cells are finer by a constant refinement factor k. As the refinement factor must adhere to the grid-structure, we use an integral multiple of the number of children created within recursive refinement. I.e., for quadtrees and octrees, we have $k = 2 \cdot \ell$, $\ell \in \mathbb{N}$ (in the special case of a 2:1 balanced octree $\ell = 1$).

Interpolation-based LBM schemes first calculate the values on the coarse grid. These values serve as boundary conditions on the fine grid by second order interpolation in space and time. This allows calculating k steps on the fine grid subsequently, whose values are then restricted back onto the coarse grid.

These schemes are widely used [161–166]. The main issue with these schemes is that they violate the Nyquist-Shannon sampling theorem [167] when populations are transferred from the fine to the coarse grid without (suitable) interpolation[1]. Without correcting the step accordingly [165, 166], these methods are not applicable to high Reynolds number flows [138].

The LBM populations are positioned on the grid points, i.e., the corners of the underlying grid. In ESPResSo's implementations, however, populations reside in cell centers. For that reason, we have not investigated these schemes further.

4.1.2 Volumetric Schemes

In volumetric LBM schemes [168–170], the LBM populations are positioned in the cell centers. This staggering of cells acts as an implicit low-pass filter and lets volumetric schemes comply with the Nyquist-Shannon theorem without further ado.

We decide to use the same scheme as PEANO [171] and waLBerla [54, 55], that is the scheme proposed by Rohde et al. [170]. The scheme is independent of the respective collision operator and first order accurate. To obtain second order accuracy, we need an additional interpolation step, e.g., *compact interpolation* [59, 172, 173]. We have not implemented this interpolation yet. Communication between different grid levels takes place through an overlap region of virtual cells that are embedded into coarse cells at refinement boundaries. The idea of virtual cells is illustrated in Fig. 4.1 for an exemplary grid section with one refinement boundary. The embedded virtual cells make the refinement boundary transparent to the streaming and bounce back step. They allow executing collision and streaming steps in locally regular grids for each level. Information traveling from a coarser cell to finer cells is first interpolated to the respective virtual cells and then streamed. Vice versa, information traveling from fine cells to a coarser neighbor is first streamed into the virtual children of the coarse cell before it is restricted to the coarse cell.

The local time step of a cell is proportional to its local mesh width *(acoustic scaling)*. We refer to this kind of multivariate time stepping as *subcycling*. Subcycling resembles a "W-cycle", see Fig. 4.2. Moreover, numerical quantities, such as relaxation parameters or external forces must be rescaled according to the respective refinement level to ensure homogeneous fluid properties across all refinement levels [1, 170].

The full algorithm is illustrated in Fig. 4.3. It begins by colliding on all available levels. Populations in virtual cells do not perform a collision step. Instead, virtual cells' populations are interpolated from the real coarse parent grid cell after its collision. In the scheme of Rohde et al., we copy populations from the parent cell to its virtual children

[1]The theorem states that the reduced signal must not contain frequencies that cannot be represented on the coarse grid side.

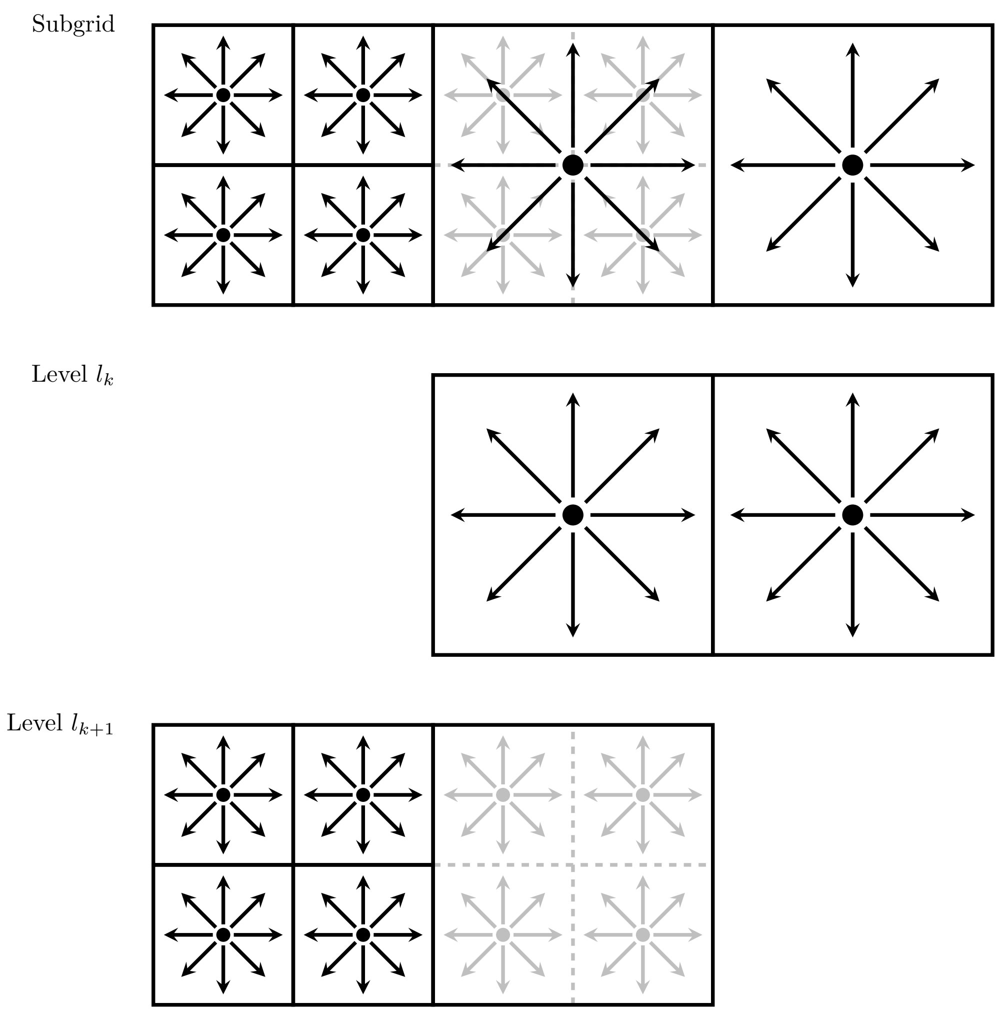


FIGURE 4.1 Top: Exemplary subdomain containg a refinement boundary. The coarse cell of level ℓ_k at this boundary embeds virtual cells of level ℓ_{k+1}, printed in gray. This leads to locally regular grids, shown in the local views of levels ℓ_k and level ℓ_{k+1} which we print in the middle and bottom. These local views are used to perform the streaming step for both refinement levels.

given a mass-free representation. All cells, i.e., real and virtual cells, of a given level participate in the following streaming step, starting with the finest level. Due to subcycling, finer cells perform several LBM steps depending on the refinement ratio. In case of 2:1 balanced grids, fine cells at refinement boundaries perform two LBM steps during one LBM step of the coarse cell, Fig. 4.2 Fig. 4.3. After streaming, populations from virtual subcells are restricted to the coarse parent cell. In the scheme of Rohde et al., we calculate the arithmetic mean of the populations in the virtual cell for each distinct velocity $\boldsymbol{c}_i$ from the stencil.

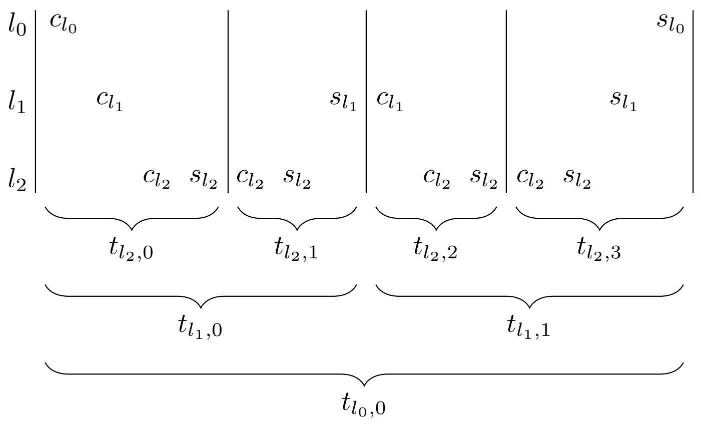

l_0
l_1
l_2
c_{l_0}
c_{l_1}
s_{l_1}
c_{l_1}
s_{l_1}
s_{l_0}
c_{l_2} s_{l_2} c_{l_2} s_{l_2} c_{l_2} s_{l_2} c_{l_2} s_{l_2}
$t_{l_2,0}$ $t_{l_2,1}$ $t_{l_2,2}$ $t_{l_2,3}$
$t_{l_1,0}$ $t_{l_1,1}$
$t_{l_0,0}$

FIGURE 4.2 We illustrate that multivariate time stepping in adaptive LBM simulations resembles a "W-cycle" by plotting the execution order of collision and streaming over three levels. Brackets indicate the size of a time step on each level. We abbreviate "collision on level i" with c_{ℓ_i} and "streaming on level j" with s_{ℓ_j}.

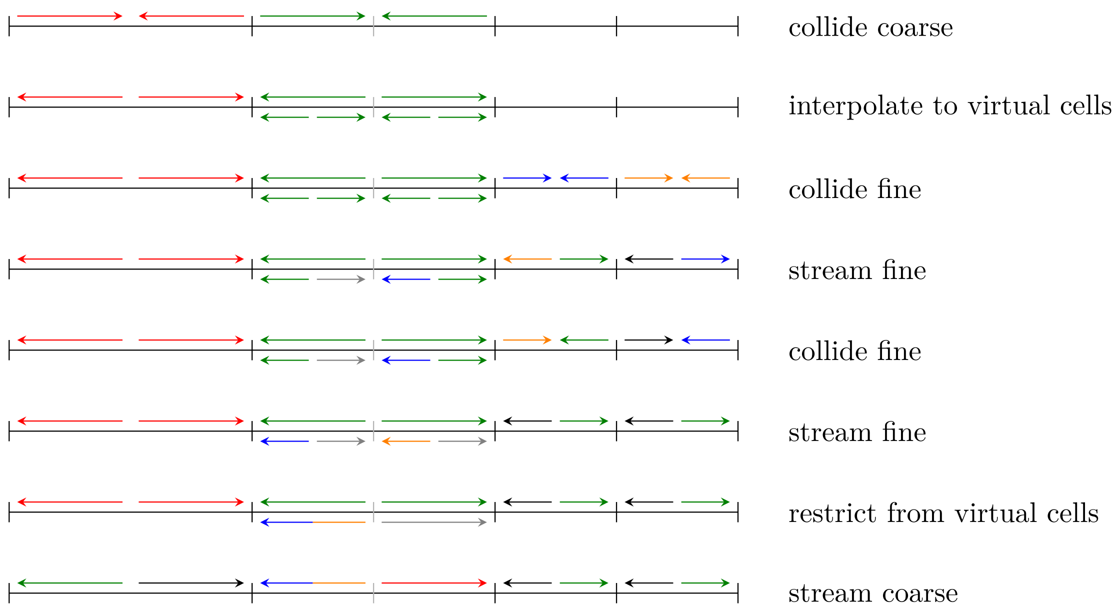


FIGURE 4.3 Schematic illustration of a coarse time step in the volumetric LBM scheme according to Rohde et al. [170]. The time axis of the figure goes from top to bottom. Different colors indicate the position of distributions at the beginning of the coarse time step. Black and gray populations are undefined: black populations stem from cells outside the Figure's domain and gray populations are algorithmically undefined within a substep. Gray populations must not propagate through the domain. The scheme works as follows: 1. Collision on coarse grid. 2. Interpolate data to virtual children of coarse cells. 3. Collide on fine grid (not on virtual fine grid). 4. Streaming step on virtual fine and fine grid. 5., 6. Repeat steps 3 and 4. 7. Restrict populations from virtual fine to coarse grid. 8. Stream on coarse grid. If the grid was not 2:1 balanced step 5 would contain more than one repetition of steps 3 and 4.

4.1.3 Data-Dependencies for Volumetric LBM

As described above, the LBM algorithm considers the adaptive grid as a set of locally regular grids. To transmit information between cells of different refinement levels, the algorithm extends the size of the fine grid by virtual cells overlapping coarse cells at refinement boundaries as illustrated in Fig. 4.1. The algorithm requires streaming data to and from real cells, leading to complex data-dependencies involving virtual cells. In the following, we address this by analyzing those data-dependencies between virtual cells. In other words, we investigate if we can optimize the implementation of the algorithm by omitting redundant streaming operations between virtual cells at refinement boundaries.

We consider an arbitrary refinement boundary in a simulation that runs on a single processor based on our definition of hanging entities from Sec. 2.1.1. Serial execution leads to an omniscient process regarding the discretization of the simulation domain, thus we avoid introducing additional complexity through different local perspectives on the grid. By definition, there is at least one coarse cell of level ℓ_k adjacent to at least one fine cell of level ℓ_{k+1}. This introduces virtual cells of level ℓ_{k+1} in all cells of level ℓ_k adjacent to that refinement boundary.

The key aspect for answering the question of redundancy for a given streaming operation between two virtual cells of level ℓ_{k+1} is the order of streaming operations at different refinement levels. As illustrated in Fig. 4.2, cells of level ℓ_{k+1}, i.e., virtual cells and fine cells, stream information before the virtual cells' host cell or cells with level ℓ_k do.

We illustrate the streaming operation for an exemplary cell in Fig. 4.4. We observe that the direction of the virtual cells' streaming operation relative to the position of the refinement boundary determines if data streamed on level ℓ_{k+1} gets overwritten by data streamed on level ℓ_k. If streaming on level ℓ_k overwrites the streamed data of its virtual subcells, we refer to the respective streaming operation between virtual cells as *optional streaming* in a specific direction, $s^{\mathrm{o}}_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i)$. Otherwise, we speak of *mandatory streaming*, $s^{\mathrm{m}}_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i)$. We recite all two-dimensional cases occurring for 2:1 balanced grid listed in [3] in Fig. 4.5. Their extension to three dimensions and the respective 2D equivalent is recited in Fig. 4.6 and Tab. 4.1.

We observe that optional streaming occurs if and only if the respective virtual cell data neither stem from nor arrive in a real cell of level ℓ_{k+1} within two streaming steps of level ℓ_{k+1}. Thus, in order to abstain from performing optional streaming steps, we generally have to perform four neighbor searches. That is, two consecutive neighbor searches in the direction of the streaming operation and in the opposite direction. Thus, we cannot answer this question locally, i.e., based on the local cell and its directly adjacent neighbors. Moreover, there are several cases that would lead to a complex code structure.

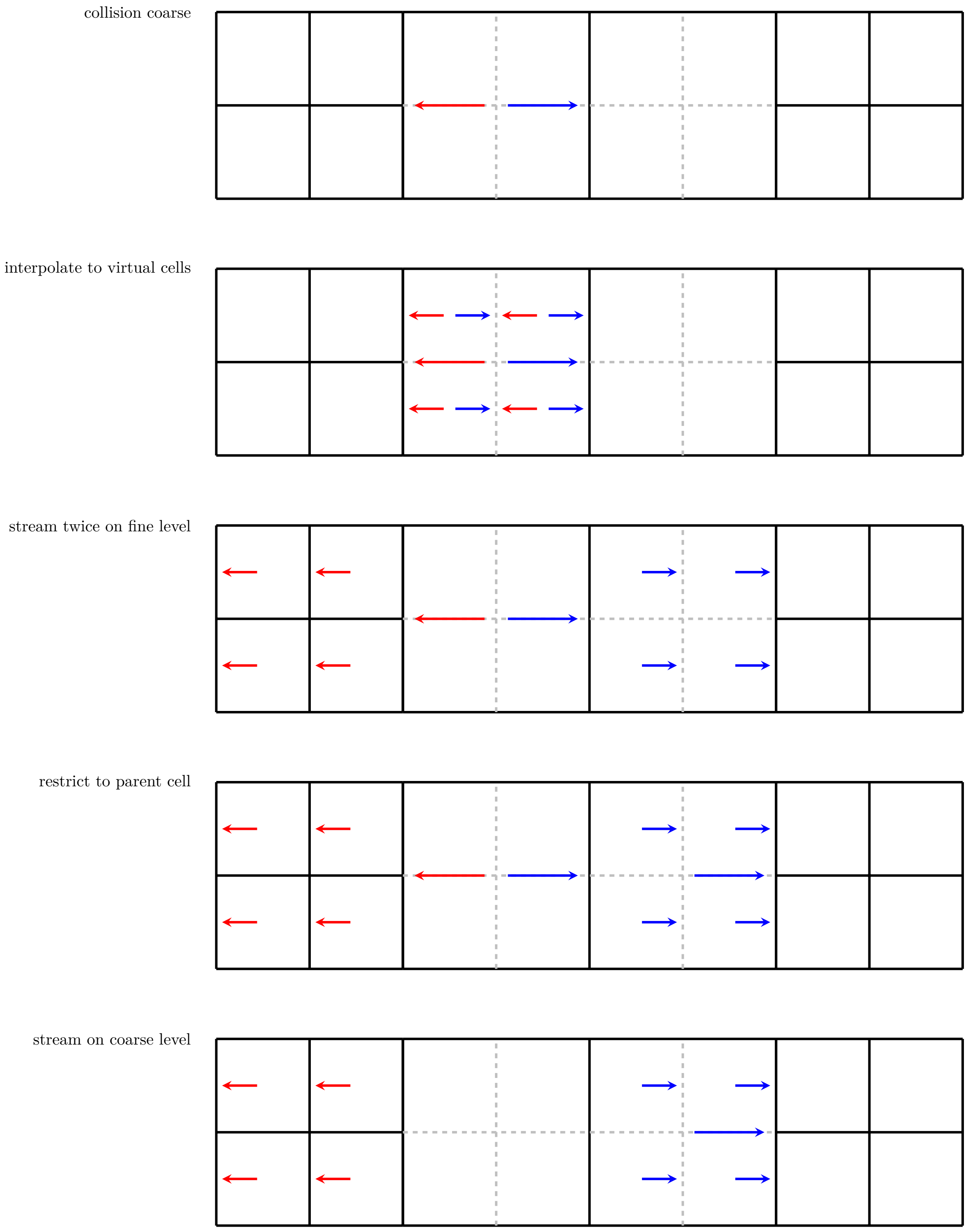


FIGURE 4.4 We trace two populations over one coarse time step at a refinement boundary. We see that if we omit streaming of the red populations, there is no way that this information reaches the fine grid cells on the left. For the blue populations, however, we see that it is irrelevant if we stream on the level of virtual cells, because that data is overwritten during the coarse streaming step.

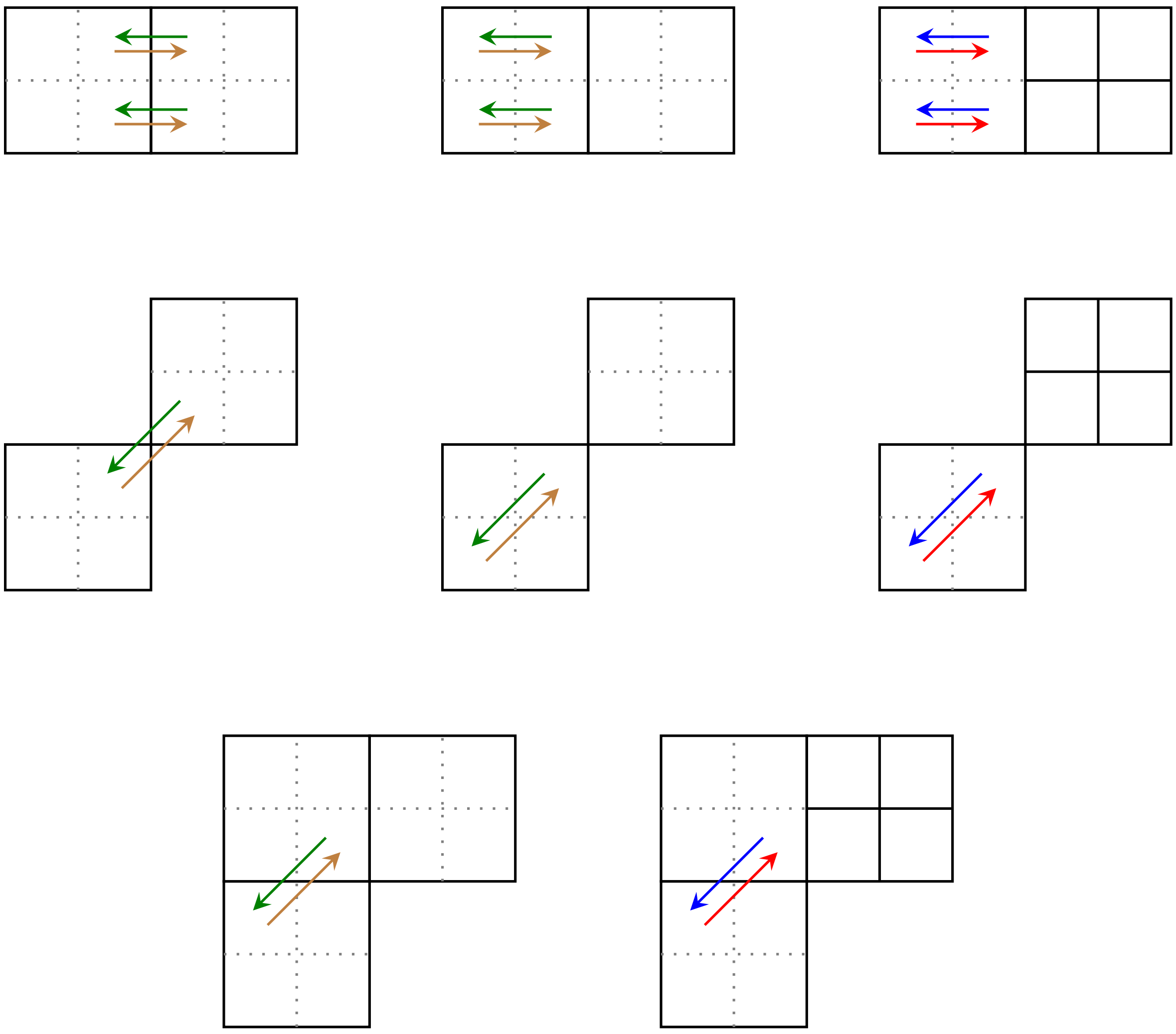


FIGURE 4.5 Two-dimensional data-dependencies for streaming between virtual cells in a 2:1 balanced grid. Arrows indicate streaming directions across the respective entities. Red and blue arrows indicate mandatory streaming operations while green and brown arrows indicate optional streaming operations. The result of optional streaming is overwritten by the following streaming step of the parent cell. We distinguish three different cases: (i) Streaming across faces (top), (ii) streaming across corners (center), and (iii) streaming across hanging corners. In each case we see that streaming is mandatory if data reach real fine cells within at most two fine time steps.

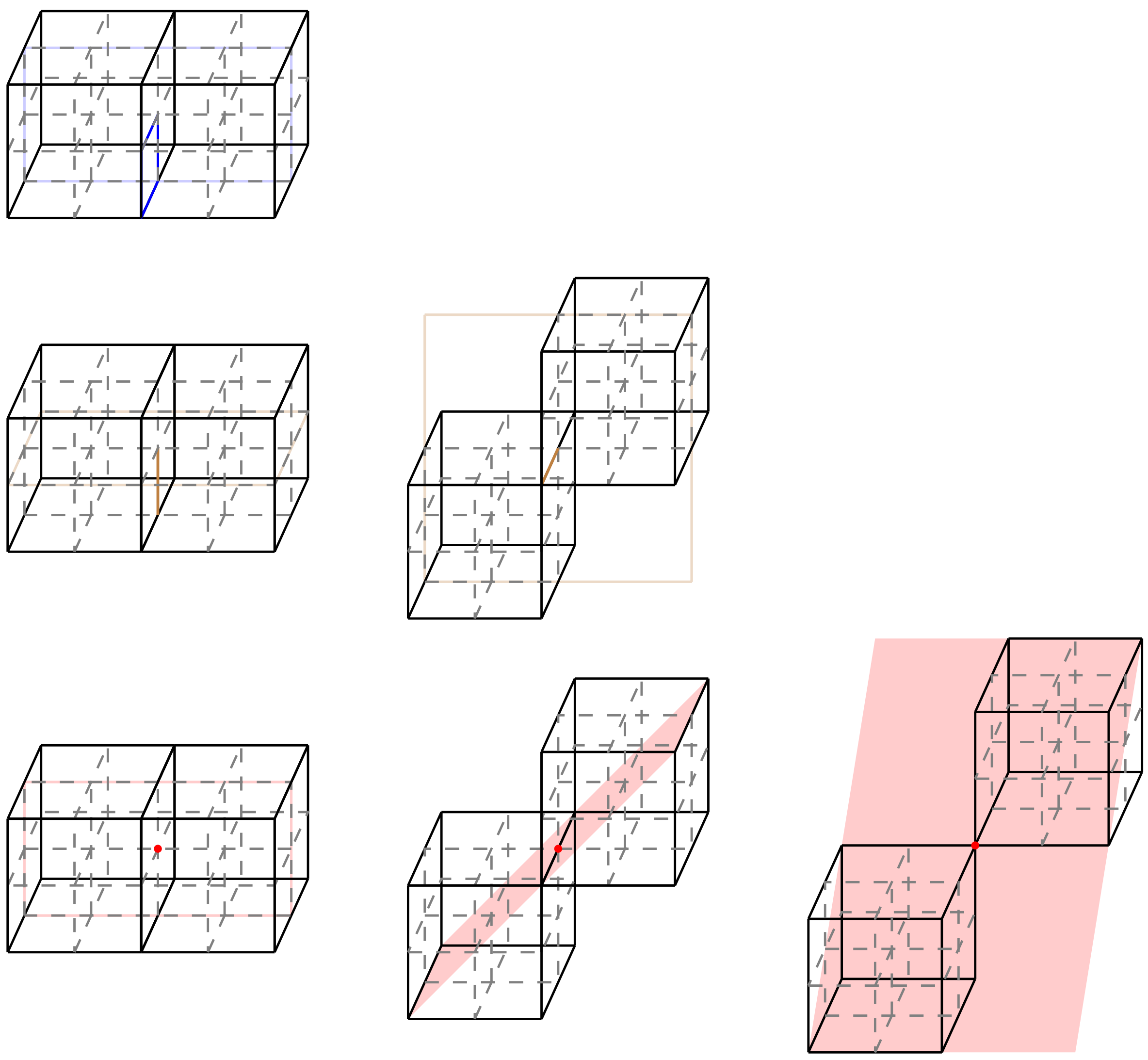


FIGURE 4.6 Three-dimensional data-dependencies for streaming between virtual cells and the projection to the respective 2D case. We illustrate the respective projections to the 2D cases by the sketched planes. The figure is organized tabularly. In the top row, we depict faces, in the middle row edges, and in the bottom row corners. In the left column we illustrate face-hanging entities, in the center column edge-hanging entities and in the right column non-hanging corners.

TABLE 4.1 Projections to reduce 3D hanging entity cases to 2D.

3D entity case	Equivalent 2D entity case
face	**face**
face-hanging face	face-hanging face
edge	**corner**
face-hanging edge	face-hanging corner
edge-hanging edge	non-hanging corner
corner	**corner**
face-hanging corner	face-hanging corner
edge-hanging corner	face-hanging corner
non-hanging corner	non-hanging corner

Thus, we do not implement our local LBM scheme in such a way that it omits optional streaming, i.e., we do not only stream $s^{\mathrm{m}}_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i)$. Instead, we stream both optional and mandatory cases, i.e., $s_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i) = s^{\mathrm{o}}_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i) \bigcup s^{\mathrm{m}}_{\mathrm{virt}}(\ell_{k+1}, \boldsymbol{c}_i)$.

While we cannot profit from omitting virtual streaming in the serial case, it, however, turns out to be helpful in the parallel case. A natural constraint to any parallel algorithm is that the number of processes may only affect the time-to-solution but not the actual result of the computation. We analyzed before, that we can either do or omit optional streaming without affecting the result of the computation. In the serial case, we found it beneficial to include optional streaming, in the parallel case, we restrict ourselves to mandatory streaming. As a consequence, we create virtual cells locally on each process, i.e., based on the information about refined regions in the respective partition and its ghost layer. In particular, no information about virtual cells induced by refinement in other partitions' inner domain is required. Moreover, the latter would substantially increase the cost of communication. We illustrate the idea in Fig. 4.7 using the same grid and the same number of processes as in Fig. 2.7. We show the three local views of each processor. Each local domain is filled by the respective color, and the hatched area indicates the part of the simulation domain of which the respective processor is aware of the discretization by its ghost layer. We frame cells where each rank locally places virtual cells based on its local information. Specifically, we place virtual cells in all ghost cells adjacent to local cells with higher level. The union of local cells leads to the global grid. In our case, we have a grid with a refined region refinement in the center. This region is surrounded by two layers of coarser cells. Thus, on a global perspective, each cell with level $\ell < \ell_{\max}$ has to host virtual cells. Going back to the local perspective of each processor, we see that this is true for all process-local cells. However, there are cell replicas in the ghost layer with levels $\ell < \ell_{\max}$ where this does not hold, e.g., ghost cells 0, 4, or 5 on the red process. We observe that none of the mentioned ghost cells with level ℓ_k receives information from local cells of level ℓ_{k+1} within two time steps of level ℓ_{k+1}. This is exactly the definition of optional streaming mentioned above.

Thus, we find an optimization that comes at zero cost but brings significant benefits. By locally creating virtual cells instead of placing virtual cells in all replicas, we omit communication during initializing the position of virtual cells. Additionally, we reduce the communication volume during run time. On top of that, this is transparent to the local implementation of the streaming algorithm, because all that happens is that local cells will not find virtual neighbor cells in a given direction for streaming. Thus, they do not execute the optional streaming step.

Up to now, we only considered data transfer between fluid cells. We extend our analysis to physical boundaries, and, again, restrict ourselves to bounce-back boundaries. We observe that virtual cells at obstacle cells are not exactly mass-conserving. The main issue is illustrated on the left of Fig. 4.8. Unlike real cells, where the complete mass is reflected back into the original cell, see Fig. 3.3, virtual cells dissipate some mass to neighboring cells. Thus, in the following coarse cell streaming step we do not overwrite the populations originating from the current cell but a combination of populations originating from different parent cells, see the right column of Fig. 4.8. Thus, virtual cells directly adjacent to obstacles cells only conserve mass up to a discretization error.

We illustrate why the error is not unbounded in Fig. 4.8 where we compare where populations originating from a given cell end up with where populations originate ending up in a given cell. We see that populations are symmetrically exchanged between cells.

We conclude that virtual cells at obstacles should be avoided. The simplest way to achieve this is to always refine boundaries up to the finest level. This also yields the best possible geometric approximation of the boundary. For the sake of completeness, we will briefly describe further adaptive LBM schemes before introducing adaptive discretizations for the ionic flux and the electrostatic potential.

4.1.4 Further Schemes

In recent years, further approaches for generalizing LBM to adaptive grids have been developed. There are approaches with variable Courant-Friedrichs-Lewy (CFL) number for octree grids [174–176].

Additionally, there are different approaches to implement LBM on unstructured grids which we briefly mention. There are spectral methods, mostly based on Discontinuous Galerkin (DG) methods [177–182]. Moreover, there is interpolation supplemented LBM (ISLBM) [183, 184]. This technique can also be used for body-fitted structured grids (moving meshes) [185]. Moreover, there are LBM schemes based on a least-squares

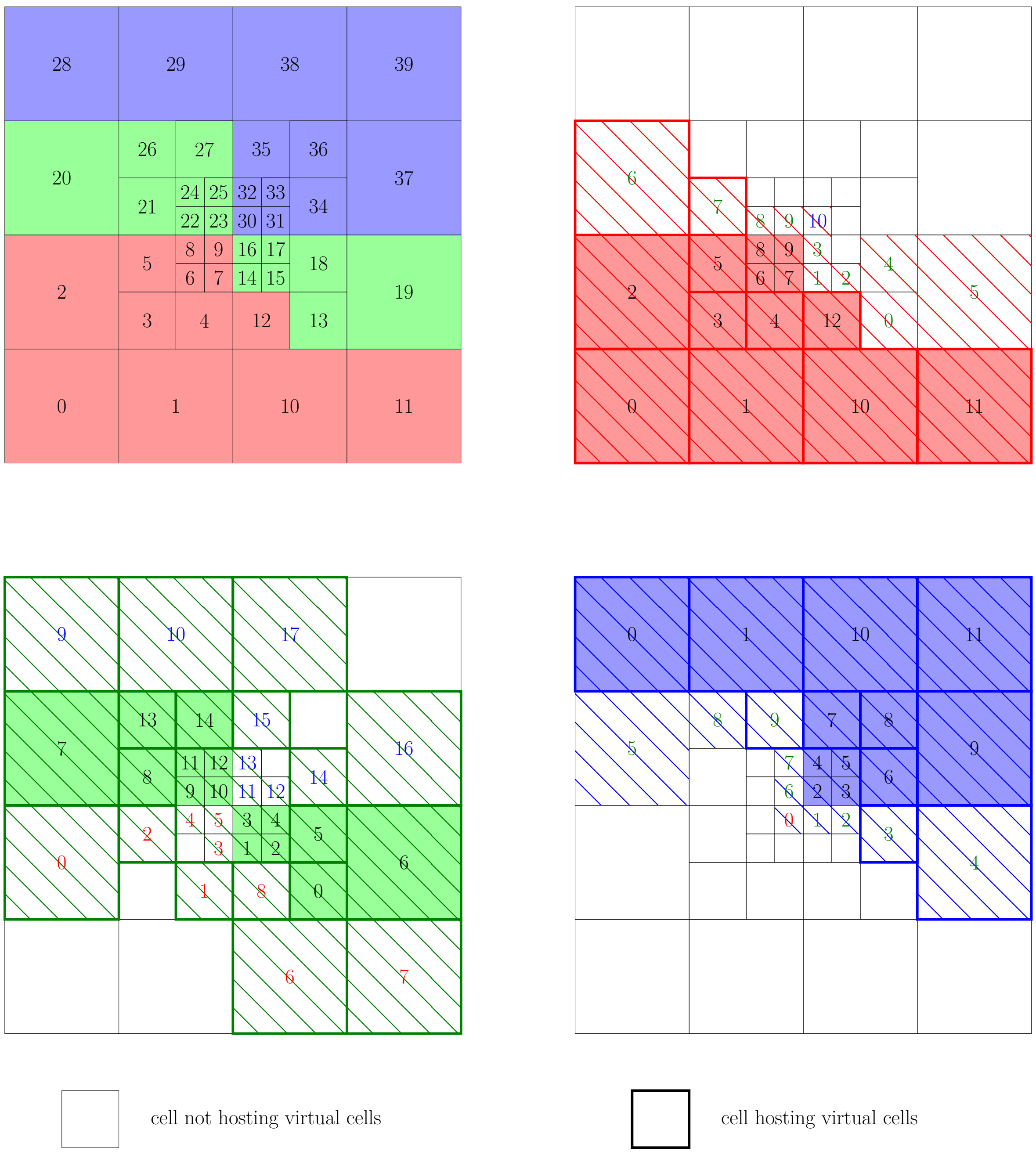


FIGURE 4.7 Example for a two-dimensional adaptive grid partitioned into three domains (top-left). We depict the three local grid views of each processor in the bottom and in the top-right corner. For each process, the local domain is the area filled by the respective color, and the hatched area indicates the ghost layer and, thus, the region of the simulation domain in which the respective processor knows the exact discretization. Local cell-indices are printed in black while we color-code ghost-indices according to the process owning the original cell. We highlight cells where each rank places virtual cells based on its local information by framing the respective cell. The union of the local perspectives yields the global perspective in the top-left corner.

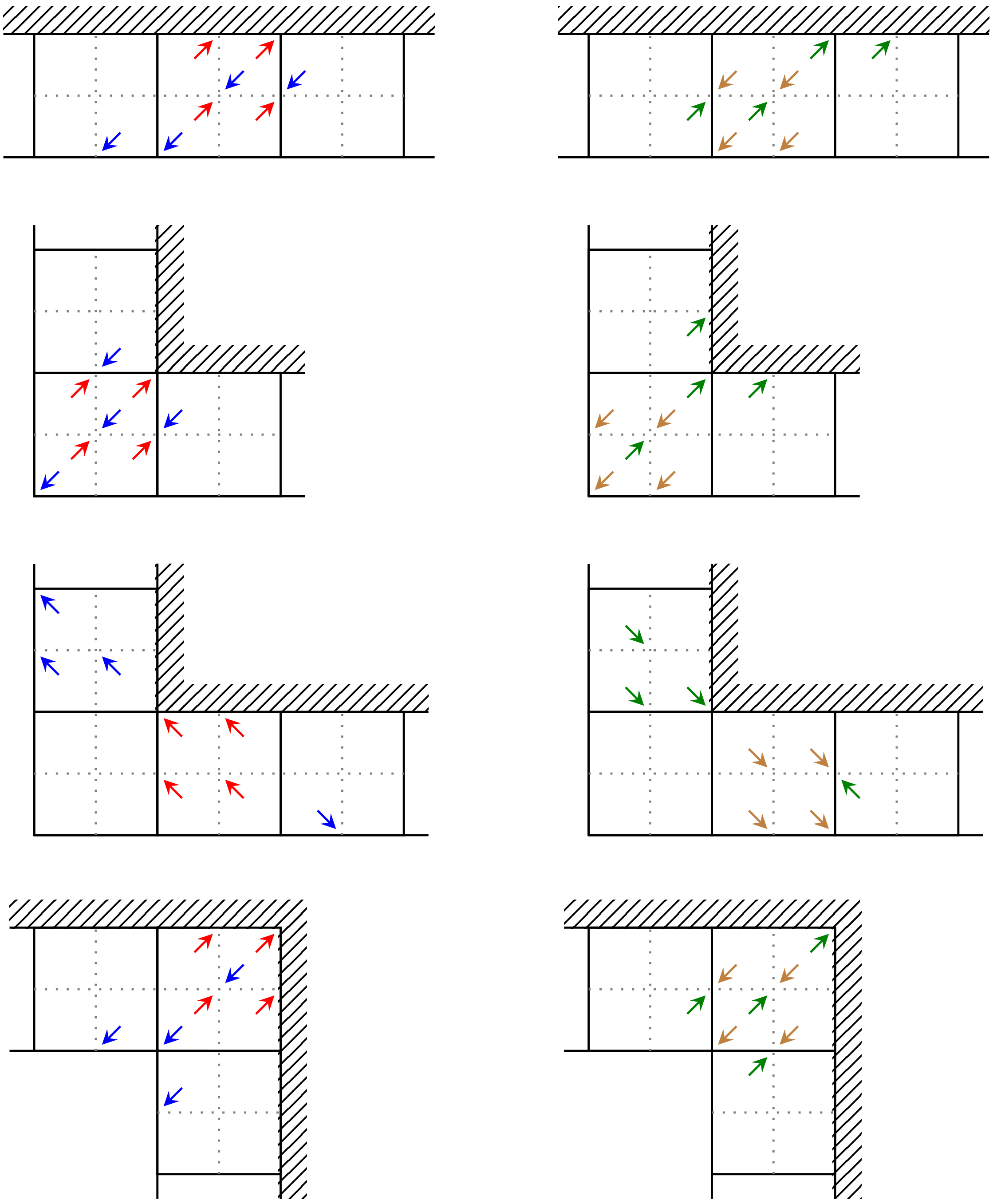


FIGURE 4.8 Bounce-back for virtual cells at outer domain boundaries. We compare where populations originating from a family of virtual cells (red arrows) end up after two streaming steps on that respective level (blue arrows) (left column) to where populations ending up in the same family of virtual cells (brown arrows) originate (green arrows) (right column). In contrast to bounce back boundaries of real cells or a regular grid (Fig. 3.3), some populations have left the original cell after the coarse time step. We observe that populations get locally exchanged between the same neighboring cells, i.e., there is a symmetrical exchange between two different parent cells.

finite element method (FEM) [186], Total Variation Diminishing (TVD) LBM based on a finite volume method (FVM) [187], or finite difference method (FDM) based LBM using short-characteristic upwinding techniques [188].

4.2 Ionic Flux

In the following, we introduce an adaptive discretization for the diffusion-advection equation (3.2) for the ionic flux from the continuous model of the electrokinetic equations introduced by Capuani et al. [118]. This discretization has been developed and integrated into ESPResSo by Ingo Tischler during his Master's thesis [5].

For both, advective and diffusive component, we use virtual cells in combination with multivariate time stepping. Similar to the lattice-Boltzmann method (LBM), this yields locally regular grids and makes the position of a refinement boundary transparent to the underlying stencil-code implementation. To transfer data between different refinement levels, we use first order interpolation. Mass-free data such as the charge density are interpolated to all 2^{dim} virtual child cells by copying the respective values. The data are restricted back to the parent cell by calculating the arithmetic mean over all virtual children. Otherwise, we interpolate data by evenly distributing mass among the 2^{dim} virtual children, e.g. local force densities. In this case, data-restriction is done by summing up data to the parent cell.

In the following, we want to discuss two more aspects of our implementation. Similar to the LBM, we have to rescale some numerical quantities. In addition to the LBM quantities described in the previous section, we have to scale the discrete diffusion coefficient according to the Einstein-Smoluchowski relation. Given a diffusion coefficient D_ℓ for level ℓ, we can derive the diffusion coefficient D_k for level k as

$$D_k = 2^{\ell-k} D_\ell. \tag{4.1}$$

Second, we know fluxes in the finite volume method (FVM) are symmetrical over shared entities. If two cells a and b touch across an entity, the inflow across that entity in cell a equals the outflow of cell b. Thus, it suffices to search half the neighbors according to a fixed pattern and use the obtained flux for both cells with opposite algebraic sign. The idea is similar to the half shell neighborhood introduced in Sec. 3.6.

There are, however, several more ways to extend the FVM to adaptive grids. We refer to, e.g., [189–192].

4.3 Electrostatic Potential

The original implementation for regular grids uses discrete Fourier transform (DFT) for discretizing the Poisson equation of the electrostatic potential, Eq. (3.3). DFT is a global transformation method where we cannot add virtual cells and treat the grid as locally regular. Thus, we choose a different approach than in Sec. 4.1 and Sec. 4.2. We use successive over-relaxation (SOR) as in the original implementation by Capuani et al. [118]. This discretization has been developed and integrated into ESPResSo by Ingo Tischler during his Master's thesis [5].

To create the stencil, we use a tensor-product approach using central differences. We deal with different refinement levels by adapting the stencil such that the respective coefficients vanish per dimension. Given a 2:1 balanced grid, there are five cases to distinguish per dimension, depending on the size of the neighbors: (i) Both neighbors have the same size. Here, we do not have to modify the stencil and can stick to the classical $(1, -2, 1)$ stencil per dimension. (ii) One neighbor has the same size, one is finer. (iii) Both neighbors are finer. (iv) One neighbor has the same size, one is coarser. (v) One neighbor is finer, the other coarser. In the cases (ii) to (v), we adapt the weights and rescale the size of the stencil such that the sum of the coefficients vanishes. We illustrate all five occurring cases for one direction in Fig. 4.9.

Alternative approaches for extending the finite difference method (FDM) to adaptive grids have been proposed in [193–197].

4.4 Molecular Dynamics

Multiple molecular dynamics (MD) codes using space-filling curve (SFC)-based domain decompositions have been developed, e.g., [198, 199]. SFC-based domain decompositions yield an efficient partitioning scheme for stencil-like algorithms [200, 201]. Adaptive variants, however, have not consistently proven to be beneficial [202–204], because of the more involved neighbor search. Additionally, in a short-range MD simulation, the main portion of the computational load is generated by the number of force pairs and only loosely coupled to the number of cells. This is fundamentally different from grid-based methods where the computational load is directly related to the number of cells. To this end, we abstain from extending the MD discretization to adaptive grids. Instead, we only provide a new SFC-based domain decomposition using `p4est` [4, 205] using regular grids.

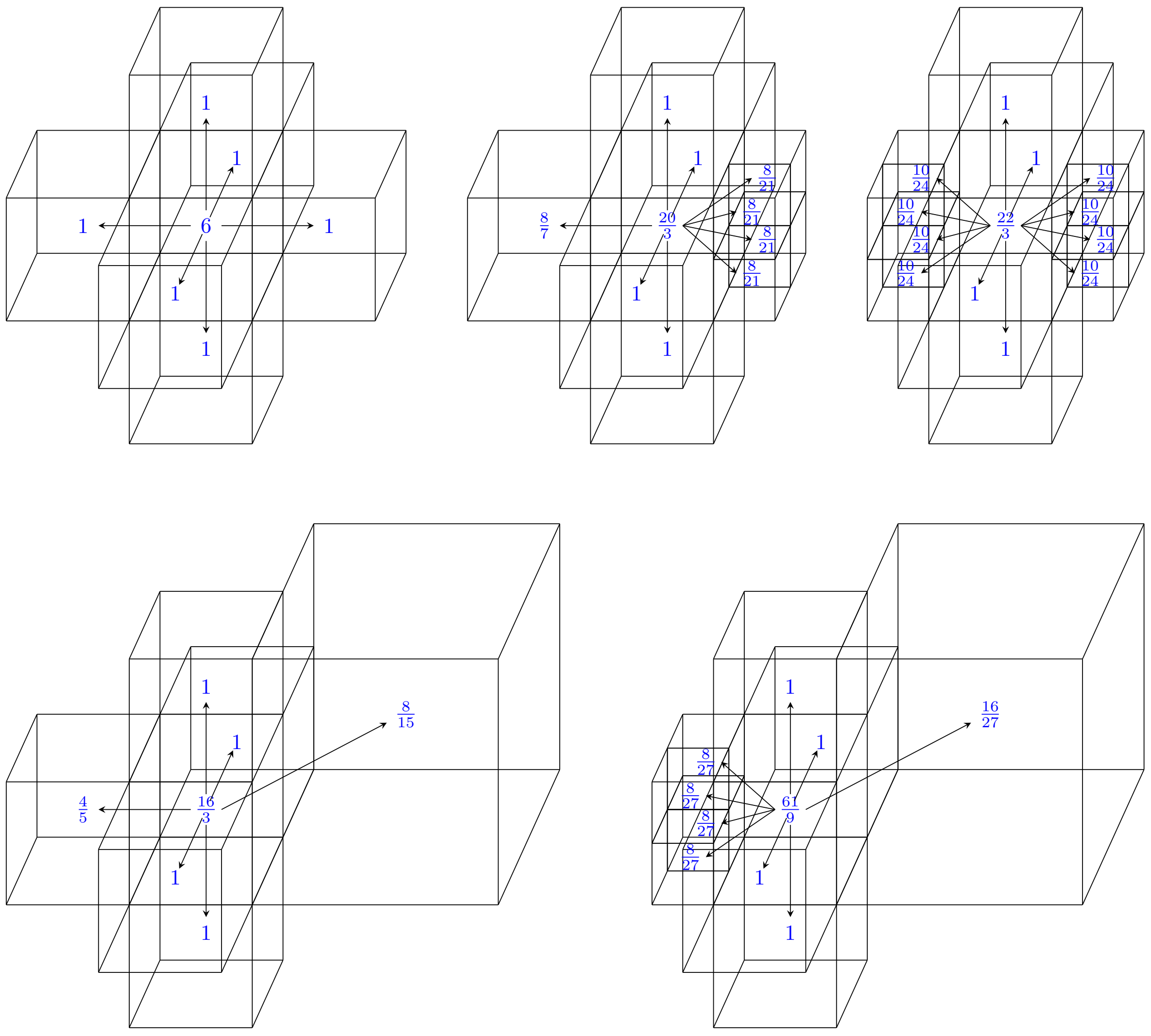


FIGURE 4.9 Exemplary FDM stencils for solving for the electrostatic potentials for all five occuring cases in the x-direction. Numbers denote the respective cell weights.

4.5 Coupling

For coupling particles and fluid, ESPResSo uses tri-linear interpolation between the eight closest fluid cells and the particle. To port the coupling to adaptive grids, we have to allow a variable number of neighbors, ranging from five (one face neighbor is same-sized, all others are double-sized) to twenty (all face and edge neighbors are half-sized). The generalized coupling has been implemented by Malte Brunn in his Master's thesis [4].

We have two constraints on the scheme. First, we want the interpolation scheme to be consistent with tri-linear interpolation in the regular case. Second, we want the scheme to be continuous across cell boundaries. I.e., if the particle changes its position, and we interpolate with different cells, we do not want the interpolant to be discontinuous. We interpolate the fluid's velocity $\boldsymbol{u}$ at the position of particle p_i as weighted sum over the fluid velocity of the respective set of neighbor cells $\{\ell_n\}$,

$$\boldsymbol{u}(p_i) = \sum_{k=0}^{|\{\ell_n\}|!} \lambda_k \ell_k \tag{4.2}$$

where λ_k is a suitable coefficient for cell ℓ_k. We calculate $\lambda_k = \alpha\beta\gamma\hat{\lambda_k}$ based on directional coefficients α, β, γ which encode the position of the particle relative to the position of the velocity and $\hat{\lambda_k}$ which encodes the number of neighbors found in a given direction. For four neighbors across a hanging face, we obtain $\hat{\lambda_k} = 0.25$, similarly we obtain for two neighbors across a hanging edge $\hat{\lambda_k} = 0.5$, and for exactly one neighbor $\hat{\lambda_k} = 1$.

This reduces the accuracy from $\mathcal{O}(h^2)$ to $\mathcal{O}(h)$. Given the one-dimensional formula for linear interpolation between two points p_i, p_j

$$f(p_i + \lambda h) = (1 - \lambda) f(p_i) + \lambda f(p_j) + \mathcal{O}(h^2), \tag{4.3}$$

we obtain positions p_j with variable distance h. For double-sized cells the actual distance between interpolation points is $1.5 \cdot h$ compared to same-sized cells, for half-sized cells $0.75 \cdot h$.

4.6 Summary

In this chapter, we have presented ways for extending the discretization of the physical models described in Chap. 3 to adaptive grids.

In the following, we describe the necessary steps to successfully integrate the above models into ESPResSo using `p4est` and present numerical results. This includes extending `p4est`, by, e.g., implementing random-access to the nearest neighbors or virtual cells. In ESPResSo, we replace the discretization of each relevant component, implement coupling, and develop criteria and algorithms for dynamically adapting the grid.

5 Integrating ESPResSo with `p4est`

We have introduced different physical algorithms and their adaptive discretization as well as algorithms for dynamically-adaptive Cartesian grids. To actually integrate ESPResSo with `p4est`, i.e., to provide a `p4est`-based grid implementation in ESPResSo, several steps have to be performed in both software packages. Regarding the `p4est` side, we provide random-access for arbitrarily connected trees and add support for virtual cells including routines for neighbor-search and ghost-exchange. We integrate our extended version of `p4est` into ESPResSo by changing the data-storage patterns and adapting kernels accordingly. We port each algorithmic component [1, 2, 4, 5, 205, 206] and couple them [4, 5, 206]. To this end, we implement a way to ensure local coupling, i.e., to align partition boundaries of different `p4est` grid-instances [207]. Additionally, we add means for dynamically adapting the grid during run time [3]. Dynamic grid-adaptivity and dynamic repartitioning are integrated with each other [205, 207].

5.1 Preparing `p4est` for Minimally-Invasive Integration

Our requirements on a grid-implementation are manifold. On the one hand, we define a basic interface of grid-algorithms in Sec. 2.1.3. As we explain in Sec. 2.2, providing efficient implementations for these algorithms in a well-designed and encapsulated form is one of our main motivations for choosing `p4est`.

We obtain more requirements from the discretization schemes presented in Chap. 3 and Chap. 4. From the demand of a minimally invasive integration, i.e., leaving as much code untouched as possible and reusing as much of the existing code as possible, we derive the need for random-access to evaluate stencil algorithms. Additionally, our grid has to support virtual cells due to the adaptive formulations of hydrodynamics, Sec. 4.1, and ionic flux, Sec. 4.2.

5.1.1 Random-Access to Direct Neighbors

For providing random-access, we enforce a 2:1 balanced grid. This significantly reduces complexity, because for each cell, each neighbor cell must either be double-sized, same-sized, or half-sized (in terms of side length). Otherwise, we would have to deal with a larger recursion depth that is only bounded by the maximum discretization level supported by the grid library.

From the alternatives for accessing neighbor data presented in Sec. 2.1.2, we choose $\mathcal{O}(1)$ lookups using neighbor lists. We store neighbor information based on the local discretization and the local ghost layer in `p4est_mesh`. We create `p4est_mesh` by traversing the grid once using `p4est_iterate` and logging the respective neighbor relations in `p4est_mesh`'s lookup tables. Besides storing neighbor relations, `p4est_mesh` can optionally create per-level lists of cells and tag cells at the parallel boundary. The lookup tables in `p4est_mesh` logically form a directed graph connecting each local cell to all of its direct neighbors. The project started with an implementation of `p4est_mesh` that did not contain neighbor relations of edges and did not fully resolve inter-tree neighbor relations across corners.

As `p4est` allows combining trees in different ways, tree-local coordinate systems may have different local orientations. Thus, it is not sufficient to only store the neighboring cells' index and calculate the neighboring entity that is connected to the current entity as

$$f_j = f_i \operatorname{xor} 1, \quad e_j = e_i \operatorname{xor} 3, \text{or} \quad c_j = c_i \operatorname{xor} \left(2^{\dim} - 1\right) \tag{5.1}$$

for faces f, edges e, and corners c according to `p4est`'s internal naming scheme, Fig. 2.6. Within a tree, however, all cells are oriented equally.

Instead, we explicitly store the neighboring cell index as well as the way the neighboring entity j is connected to the local entity i. Besides the entity indices over which the respective cells are connected, we also have to encode the size of the neighboring cell and if the cell is a local cell or a replica in the ghost layer. To encode the way that entities are connected, `p4est` has a concept of "orientation". For defining the orientation r between cells connected over a face, we use face corners ξ_i, that is numbering the corner indices c_j enclosing a face f_k consecutively according to `p4est`'s naming scheme, Fig. 2.6. In Fig. 5.1, we illustrate the idea: Face f_0 is surrounded by the corners c_0, c_2, c_4, and c_6. The lowest corner index, c_0, is assigned face corner index ξ_0 and so on. The orientation is defined as follows: Given two faces f_i, f_j with index $i < j$. Then, the orientation r is determined by the face corner index ξ_k on face f_j that is touching face corner ξ_0 on face f_i. We illustrate the four different orientations for a "fixed" face f_1 on the left and a face f_0 on the right that we rotate to have the specified orientation in Fig. 5.1. Here, the

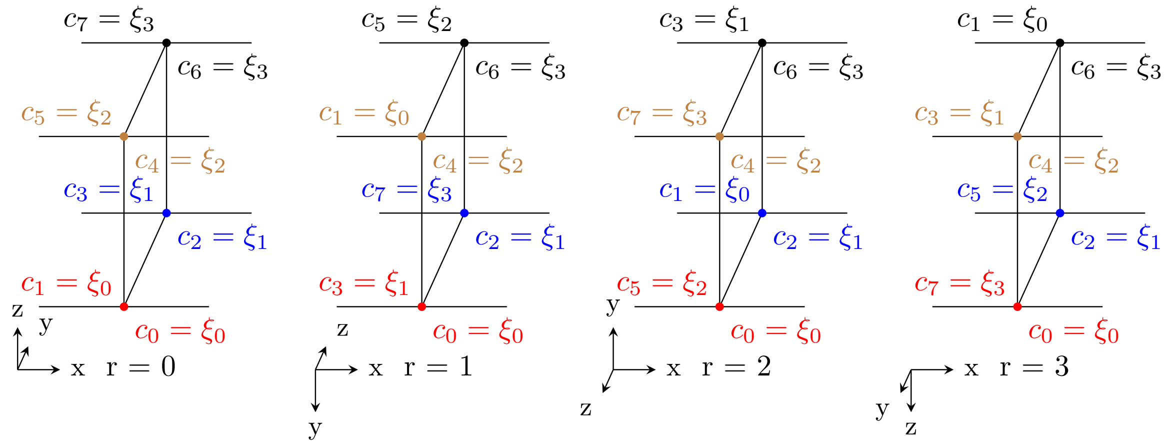


FIGURE 5.1 p4est defines relative orientation of two trees via faces. We illustrate this for the connection of the "fixed" face f_0 on the right and face f_1 on the left which we rotate to obtain the respective orientation. As $0 < 1$, the face corner touching ξ_0 on f_0, the face on the right, defines the orientation. For clarity, we color the face corners of f_0 consistent in each figure. The resulting local coordinate system of the tree on the left is printed beside the orientation. The local coordinate system of the right tree is fixed as in $r = 0$.

face corner index ξ_i of the fixed face f_1 touching face corner ξ_0 of face f_0 determines the orientation. Later we will see that we can derive a similar concept for edges.

Given the orientation r and two face indices f_i and f_j, Burstedde et al. present a transformation to map a face corner index ξ_i of f_i to the corresponding face corner index ξ_j on face f_j [85] as

$$\xi_j = \mathscr{P}_c(\mathscr{Q}_c(\mathscr{R}_c(f_i, f_j), r), \xi_i) \equiv \xi_j(\xi_i), \tag{5.2}$$

where

$$\mathscr{R}_c = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 & 1 & 0 \\ 0 & 2 & 2 & 0 & 0 & 1 \\ 0 & 2 & 2 & 0 & 0 & 1 \\ 2 & 0 & 0 & 2 & 2 & 0 \end{pmatrix} \qquad \mathscr{Q}_c = \begin{pmatrix} 1 & 2 & 5 & 6 \\ 0 & 3 & 4 & 7 \\ 0 & 4 & 3 & 7 \end{pmatrix} \qquad \mathscr{P}_c = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 1 & 3 \\ 1 & 0 & 3 & 2 \\ 1 & 3 & 0 & 2 \\ 2 & 0 & 3 & 1 \\ 2 & 3 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}. \tag{5.3}$$

The rows of $\mathscr{P}_c$ list permutations of face corner indices corresponding to the 2×4 elements of the dihedral group of face orientations. The matrices $\mathscr{R}_c$ and $\mathscr{Q}_c$ choose the correct one in a non-redundant way. We use an additional lookup table to map from a tuple of face index and face corner index to the actual corner index.

Extending this concept to edges is straight-forward, because faces are also enclosed by the same fixed edges that are, themselves, enclosed by the same corners. We developed this in [1]. Analog to face corners ξ_i, we define face edges ν_i and edge corners μ_i by numbering edge and corner indices consecutively according to `p4est`'s naming scheme.

Each face edge μ_i is enclosed by two face corners ξ_i that can be written as columns of

$$\Xi_i = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 1 & 3 & 2 & 3 \end{pmatrix}. \tag{5.4}$$

Face f_0 touches the four edges e_4, e_6, e_8, and e_{10}, thus we refer to edge e_4 as face edge ν_0. Face edge ν_0 touches face corners ξ_0 and ξ_1, ν_1 touches ξ_2 and ξ_3. Thus, the columns of Ξ_i contain for each face edge index ν_i the respective face corner indices ξ_i and ξ_j. We refer to the lower face corner index as edge corner μ_0 and to the larger face corner index as edge corner μ_1. According to this definition, we can derive the face corners ξ_i seen from the adjacent tree by applying Eq. (5.2) for both elements of the respective column of Ξ_i,

$$\Xi_j(k, \nu_i) = \xi_j\left(\Xi_i(k, \nu_i)\right), \quad k = 0, 1. \tag{5.5}$$

Using Eq. (5.4) in reverse, the resulting two face corners can be identified one-to-one with the face edge's number seen from the other tree. We condense this into the form used earlier by writing the edge transformation

$$\nu_j = \mathscr{P}_e(\mathscr{Q}_e(\mathscr{R}_e(f_i, f_j), r), \nu_i) \equiv \nu_j(\nu_i) \tag{5.6}$$

with

$$\mathscr{R}_e = \mathscr{R}_c \qquad \mathscr{Q}_e = \begin{pmatrix} 4 & 1 & 2 & 7 \\ 0 & 6 & 5 & 3 \\ 0 & 5 & 6 & 3 \end{pmatrix} \qquad \mathscr{P}_e = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 3 & 2 \\ 1 & 0 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 1 & 0 \\ 3 & 2 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}. \tag{5.7}$$

Again, $\mathscr{P}_e$ lists the relevant permutations of the index set $\{0,1,2,3\}$ which $\mathscr{R}_e$ and $\mathscr{Q}_e$ select accordingly.

For edge corners, the number of cases is smaller, because the edges either have the same orientation, i.e., edge corner μ_0 of edge e_i touches edge corner μ_0 of edge e_j or the edges are flipped. We can condense the transformation into

$$\mu_i = \mu_j \operatorname{xor} r_e \tag{5.8}$$

where r_e denotes the orientation of the edge. For edges with the same orientation, we set $r_e = 0$, for flipped edges we set $r_e = 1$.

These transformations allow finding the correct neighbor for each cell from the list of cells returned in an entity-callback issued by `p4est_iterate`. These callbacks are functions which `p4est` calls upon visiting the respective entity when traversing the grid, passing status information and the respective cells touching the respective entity as parameters. The callback function is called with a variable number of sets of cells. In case of face neighbors, the number of sets is always two; at non-periodic domain boundaries one set is empty and if the face is hanging one set contains four cells. All other sets contain exactly one cell. In this case, cells in the first set are face neighbor of the second set. For edges and corners the number of sets is generally four for edges and eight for corners. However, in case of tree boundaries, any arbitrary number of sets is possible. In case of edge neighbors, hanging sets contain two cells and in case of corner neighbors there are no sets containing more than one cell. Additionally, as we illustrate in Fig. 5.2, if we consider a cell of a specific set, not all remaining sets contain valid edge neighbors. If we, e.g., log the neighbors of a cell from the blue set, we may only log cells from the black set. Cells from the green and brown set are face neighbors to cells from the blue set. When processing the other sets, we have to filter different sets accordingly. In intra-tree cases, we can deduce the edge index of the correct neighboring set using Eq. (5.1). This allows processing the respective sets pairwise, because cells of the blue and the black set filter cells from the green and the brown set. For inter-tree cases, we transform the current edge over both neighboring faces into the respective neighboring tree. This allows determining those edge indices that are actual face neighbors. We omit these cells while logging neighbor relations. As we do not know the number of trees adjacent to an edge beforehand, we process each set separately. For corner neighbors, we filter face neighbors and edge neighbors using the same concepts. In intra-tree cases we log neighbors pair-wise while we process each set of cells separately in inter-tree cases.

Creating the aforementioned sets of cells is handled by `p4est_iterate` [86] and not part of the work in this thesis. During this thesis, we have extended `p4est_mesh` to

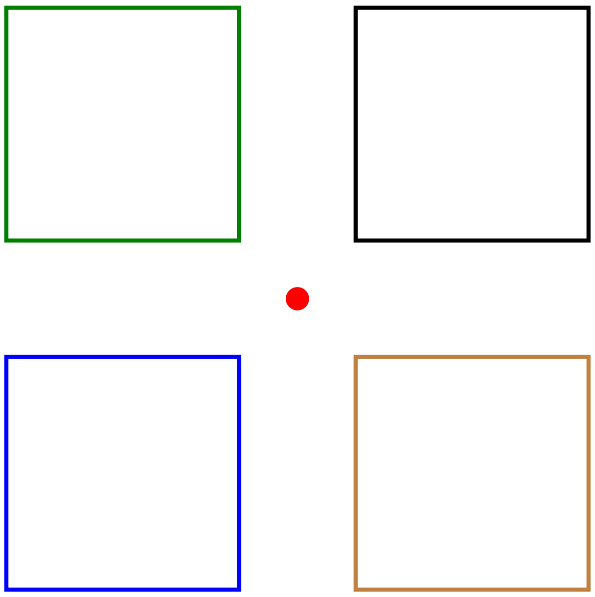


FIGURE 5.2 Top view on the sets of cells passed as parameter when entering an edge callback of `p4est_iterate`. We process all four sets in this call, thus, we loop over the sets and log cells of those sets that are not face neighbors. If we, e.g., process the blue set, we log cells contained in the black set and omit cells contained in the green and brown sets.

include edges and to correctly filter edge neighbors for inter-tree corner neighbors. To this end, we developed a storage-scheme and an encoding scheme similar to faces and corners. For inter-tree edge and corner neighbors, we use the same compressed sparse row (CSR)-like storage scheme as in `p4est_ghost`. Analog to the existing implementation for faces and corners, we encode size, orientation, and neighboring entity within one integer value. To indicate that a cell is part of the ghost layer, we add the number of local cells to the cell's ghost index, i.e., if there are eleven local cells on a domain, we assign them the cell index 0, . . . , 10 and the first ghost cell is assigned cell index 11. We have developed a neighbor-lookup function to provide the user with an easy-to-use interface that provides a uniform way for querying face, edge, and corner neighbors. This function encapsulates the internal storage scheme of `p4est_mesh` which uses separate arrays to store information about face, edge, and corner neighbors. It fetches data from the `p4est_mesh` data-structure and appends it into a set of user-defined arrays.

5.1.2 Integration of Virtual Cells

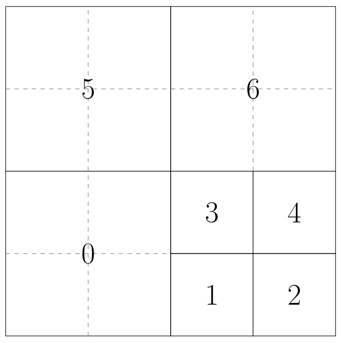
Realization in p4est. We realize virtual cells in a separate, new component in `p4est` which we call `p4est_virtual`. As `p4est` stores only leaf cells instead of the full tree structure, it does not store overlapping cells. Integrating virtual cells requires actually extending `p4est`. However, we do not want to abandon the leaf-only concept. We can integrate virtual cells in two fundamentally different ways. Either we store virtual cells explicitly by refining each cell at a refinement boundary and virtually add one real cell, or we keep the real cell in `p4est` and separately add the virtual child cells. We choose the latter solution, because it implies that each `p4est` cell is an actual grid cell. This point seems trivial, however, its implications are fundamentally important. Storing only real cells in `p4est` means that we can reuse all internal algorithms as they are. Otherwise, we would have to add conditionals to each internal algorithm — such as refinement, coarsening, 2:1 balancing, partitioning, creating the ghost layer or the lookup-tables for random access — to treat virtual cells in such a way that we recover the behavior of real cells. Thus, we minimize the risk of interfering with `p4est`'s efficiency and scaling behavior and reduce the complexity of the task at hand. With this in mind, we choose to integrate virtual cells as light-weight as possible. Our implementation of `p4est_virtual` is designed to extend `p4est_mesh` using the same design principles.

We tag each local and ghost cell hosting virtual cells. Thus, virtual cells do only exist in the payload data and as a tag, i.e., we do not create actual cells in `p4est`. To map cells in `p4est_virtual` to their data, we provide an offset, i.e., the number of cells preceding the current real cell as well as the first virtual subcell if the current cell is at a refinement boundary. Similar to `p4est_mesh`, we optionally store a per-level list of cells and an offset per-level.

This design constraints the ways of storing data. `p4est_virtual` is designed to support both layouts introduced in Sec. 2.1.2 and illustrated in Fig. 2.4. We allow storing data in a plain array where data of virtual cells are inserted in the array in Morton-order after the parent cell. Additionally, we support storing data in per-level Morton order. Here, virtual cells are inserted according to their parent's Morton index in separate lists according to the cell's refinement level. We illustrate both data schemes for an exemplary grid in Fig. 5.3 and the information that allows retrieving the data associated with the respective cell. As we use multi-variate time-stepping to discretize the electrokinetic equations introduced in Sec. 3.2, we choose to use the per-level data layout.

In both cases we have to derive the offset of the current cell to retrieve its associated data. For the plain array, we calculate the offset from the cell id and the last cell hosting virtual cells, i.e., the last flag that is not set to '-1'. For the per-level storage scheme, we directly use the offsets in real and virtual data structures in Fig. 5.3. These values index into the level arrays by yielding the number of cells preceding the current real or the first virtual cell *on their respective level*.

Following the results of our analysis in Sec. 4.1.3, we omit communication for positioning virtual cells in the ghost layer. As each process knows the full neighborhood for each of its cells, we can also locally derive for each ghost replica if the respective



plain	0	0_{v0}	0_{v1}	0_{v2}	0_{v3}	1	2	3	4	5	5_{v0}	5_{v1}	5_{v2}	5_{v3}	6	6_{v0}	6_{v1}	6_{v2}	6_{v3}

per-level

level l_1	0	5	6													
level l_2	0_{v0}	0_{v1}	0_{v2}	0_{v3}	1	2	3	4	5_{v0}	5_{v1}	5_{v2}	5_{v3}	6_{v0}	6_{v1}	6_{v2}	6_{v3}

virtual flags	0	-1	-1	-1	-1	1	2
offset level real	0	4	5	6	7	1	2
offset level virtual	0	-1	-1	-1	-1	8	12

FIGURE 5.3 We use the above grid to illustrate the supported data-layouts for real and virtual cells, plain and per-level. Additionally, we show the available information in `p4est_virtual`. For real cells without virtual cells, we set the virtual flag to '-1'. For the plain array, we obtain the offset from the cell id and the flag or the last flag that differs from '-1'. In case of the per-level storage scheme, we use the offset arrays. These values index into the level arrays and indicate the number of cells preceding the real or the first virtual cell on their respective level.

processor will place virtual cells inside this replica and adapt the local communication rules accordingly. In Sec. 4.1.3 we show that information from virtual cells in ghost replicas induced by a different partition is not required for streaming correctly.

In the following, we explain the way we detect refinement boundaries, how we build the ghost layer and include virtual cells in the data-exchange, as well as the necessary extensions for providing random-access including virtual cells.

Detecting Refinement Boundaries and Generating Virtual Cells. As we have decided to create `p4est_virtual` as a separate component, we need an additional grid traversal for embedding virtual cells at refinement boundaries. We detect refinement boundaries by traversing each cell and checking the size of its neighbors. If there is any smaller neighbor, we embed virtual cells. We use the information in `p4est_mesh` to check the size of the neighboring cells which restricts us to 2:1 balanced grids.

If the input-data in `p4est_mesh` includes tags for the parallel boundary we can optimize this process for inner cells. Using dynamic programming and early loop-exits, we can minimize the number of necessary neighbor-queries. We stop querying neighbors as soon as we know that the current cell hosts virtual cells, and we “inject” virtual cells into all double-sized neighbors we find. Otherwise, i.e., if we either do not know if a cell is adjacent to a parallel boundary or if a cell is actually adjacent to a parallel boundary, we have to query all neighbors. As explained in Sec. 5.1.1, ghost cells are no valid starting points for neighbor queries in `p4est_mesh`. Thus, we have to query all neighbors, because they could be larger ghost cells. In this case, these ghost cells must embed virtual cells on this rank.

This leaves calculating offset values. For the plain array, we obtain the offset o of cell c_i from

$$o_{\text{real}}(c_i) = n_{\text{cell, real}} + 2^{\text{dim}} n_{\text{cell, virtual}} \tag{5.9}$$

where n is the total number of real and virtual cells with cell id $c_k < c_i$. If the cell hosts virtual cells, their payload is stored directly after the parent’s cell, thus, the offset of its first virtual subcell is $o' = o + 1$. We populate the level offset values by counting the number of real and virtual cells on each level. Then, we obtain the offset o for the real cell c_i with level ℓ as

$$o_{\text{real}}(c_i) = n_{\text{cell, real}}(\ell) + 2^{\text{dim}} n_{\text{cell, virtual}}(\ell), \tag{5.10}$$

where $n(\ell)$ is the number of real and virtual cells on level ℓ with cell id $c_k < c_i$. For evaluating the offset of the first virtual cell, we use Eq. (5.10) with level $\ell' = \ell + 1$.

Ghost-Exchange Including Virtual Cells. To perform simulations in parallel, `p4est` provides functions in `p4est_ghost` that allow exchanging data. These work fine, allow communication hiding, i.e., overlapping communication and computation, and can handle different scenarios such as only transferring quadrants of a specific level. However, if a process boundary coincides with a refinement boundary, there are virtual cell data to be potentially sent and received. Moreover, Fig. 5.3 illustrates that we have offsets in the data induced by virtual cells. To address both issues, we extend `p4est_ghost` in a component we call `p4est_virtual_ghost` which allows directly integrating virtual cells in the ghost-exchange. Compared to Fig. 2.7, we add an array that indicates for each mirror cell, if the respective receiving process expects virtual cells. This is the case if that process places virtual cells in the respective cell. As explained above, this happens if and only if the current cell has neighbors with a higher refinement level that are local cells on the respective receiving process. We illustrate our extension to `p4est_ghost` in Fig. 5.4.

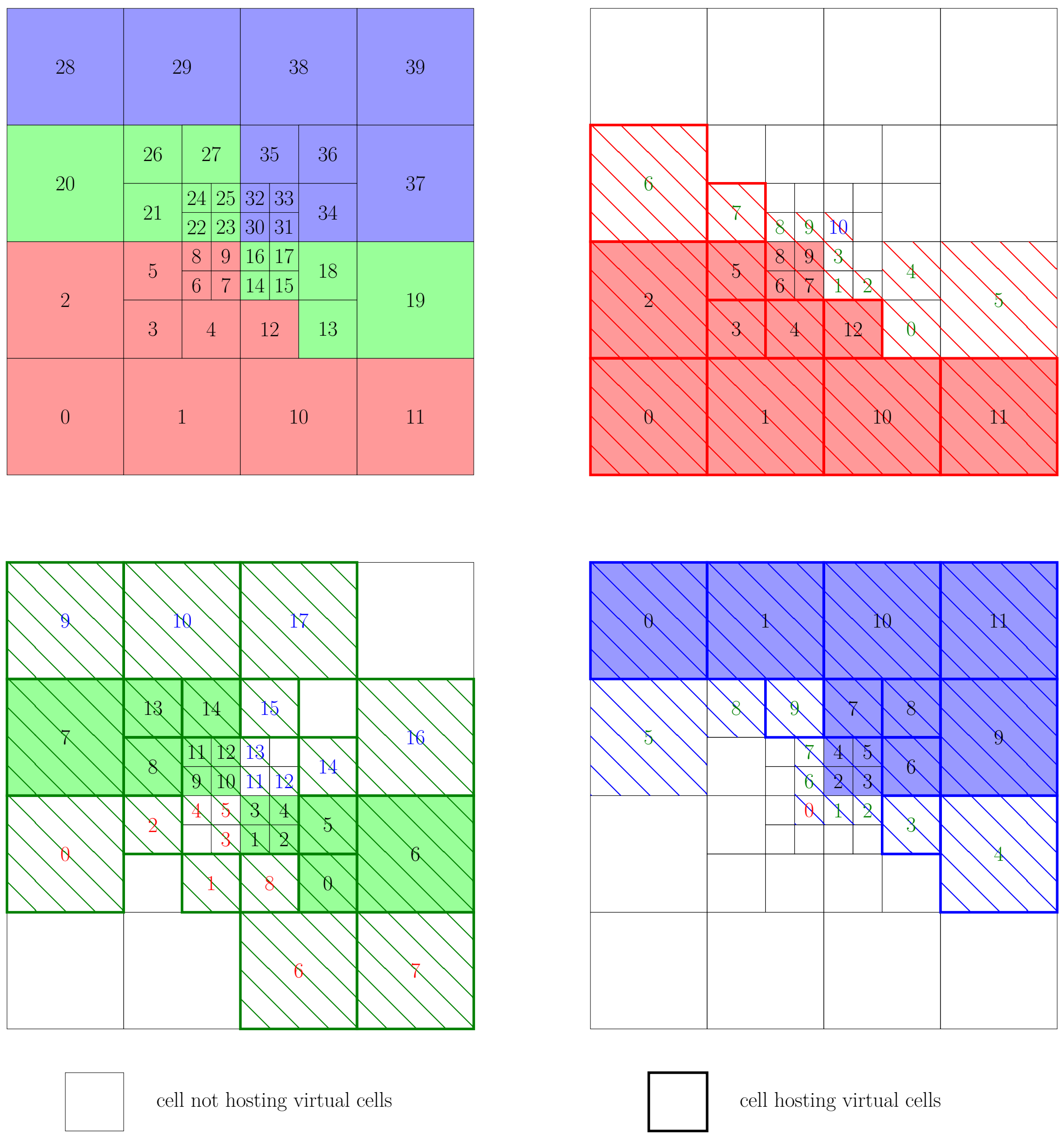
We support both storage schemes visualized in Fig. 5.3 and allow communication hiding using the same mechanisms as `p4est_ghost`. Independent of the chosen storage scheme, we iterate over the local mirrors and copy their payload in separate send buffers. We receive the data directly in the respective ghost data-structure, i.e., we avoid allocating additional receive buffers and copying or moving data.

Random-Access Including Virtual Cells. The most challenging task in integrating virtual cells is to integrate them into the random-access logic. For searching neighbors when only real cells are involved, we obtain a set of cell indices and encodings from the neighbor search using `p4est_mesh`. In intra-tree cases, the size of the set can be either one, two, or four, while in inter-tree cases, the size of the set may be k, with $k \in \mathbb{N}_0$.

Adding virtual cells, the size of the set of cells returned from neighbor search in intra-tree cases is always either one or zero. If it is one, the neighboring cell must have the same size as the current cell by design. Thus, the encoding comprises the neighboring entity and orientation, the size information is obsolete. However, cell index and encoding, are not sufficient to uniquely identify a neighboring cell, because we could either mean the real cell or one of its 2^{dim} virtual subcells. Therefore, we additionally return the virtual index of neighboring cells or '-1' if we refer to the real cell.

In the following, we explain how to find neighbors including virtual cells, i.e., how to find the three values of cell index, virtual index, and encoding. `p4est_mesh` contains information about real cells. Therefore, we have no direct way for obtaining the virtual cell index and no direct information at all when searching for neighbors of virtual cells. In the following, we briefly describe the algorithms for searching neighbors from real and virtual cells, respectively. In a 2:1 balanced grid, we have three times two cases, because real and virtual cells may search for neighbors. We illustrate the six occurring cases in Fig. 5.5.

Given a real cell, we can perform an ordinary neighbor search using `p4est_mesh` and inspect the result. If we find a same-sized neighbor we pass it through and there is no virtual cell index to be determined. We discard half-sized neighbors, because they will be dealt with at the level of the current cell's virtual subcells. For a double-sized neighbor, we must find the correct virtual cell within this neighbor to obtain a same-sized neighbor. In case of corner neighbors, the respective virtual cell is the one assigned to the current corner in the neighboring cell. In case of faces or edges, we know the neighboring cell's face or edge corner index touching the current cell's entity as well as our subcell's index and the respective orientation from the neighboring encoding. This is all we need for transforming the current cell's sub-index onto the respective face or edge, which gives the



Green processor:

ghosts	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
proc_offsets	0	9	9	18															
mirrors	0	1	2	3	4	5	6	7	8	9	10	12	13	14					
mirror_proc_mirrors	0	1	2	3	5	6	7	8	9	10	3	4	5	6	7	10	12	13	14
mirror_proc_offsets	0	10	10	19															
mirror_proc_virtual	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1

FIGURE 5.4 We illustrate the vector `mirror_proc_virtual` which extends `p4est_ghost` in `p4est_virtual_ghost`. We read the data as in `mirror_proc_mirrors`. Highlighted indices are local virtual cells not mirrored in their replica. Thus, their data are not sent during ghost exchange. The color matches the coloring of the respective receiving process's domain.

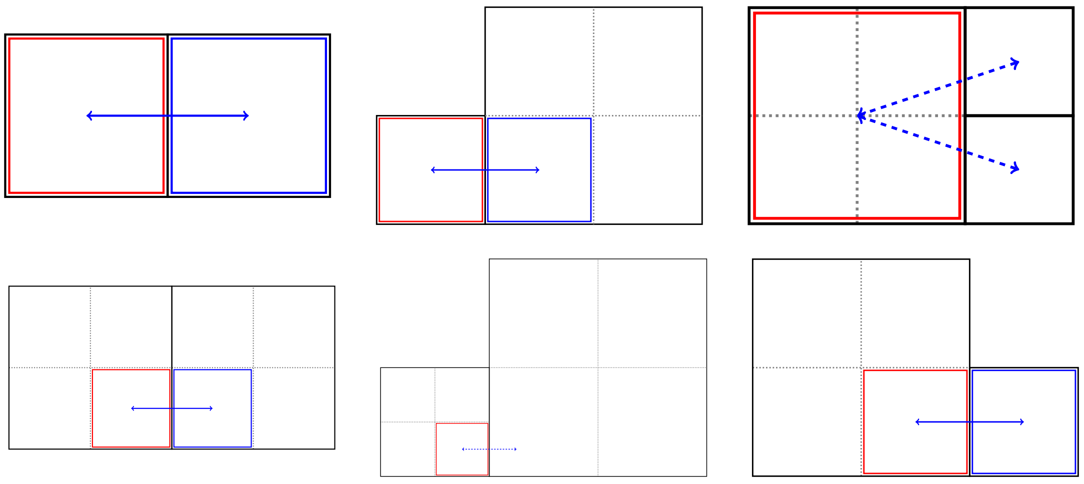


FIGURE 5.5 We illustrate all three possible cases when finding a neighbor in a 2:1 balanced grid. The search originates from the red cell and can yield same-sized, double-sized, or half-sized neighbors. In case of a real cell (top), this size difference relates to the actual cell searching for neighbors. For a virtual cell (bottom), the different size relates to the parent cell. All valid neighbor relations are marked in blue.

virtual cell index.

For neighboring queries originating from a virtual cell, there are two cases which we illustrate in Fig. 5.6: we either search for neighbors in the same host cell or outside it. Given a virtual cell and a direction that yields a neighbor in the same host cell, we already know the cell index and can easily derive the virtual cell index. We generate the missing encoding from Eq. (5.1). For a virtual cell that requires searching neighbors outside the parent cell, we always perform an indirect neighbor search, i.e., we first search the neighbor of the real host of the virtual cell and then derive the respective same-sized neighbor of the virtual cell if that neighbor exists. We adapt the direction of the neighbor search from the real host where necessary. We illustrate the different cases in Fig. 5.6 for an exemplary virtual cell that has neighbors inside its parent cell as well as outside its parent cell.

The size information we obtain in the encoding relates to the parent cell, i.e., a same-sized neighbor actually is a double-sized neighbor from the virtual cell's point of view. We do not store this information into another set of lookup tables. Instead, we re-generate the information about the vicinity of the current cell in each neighbor query.

If we obtain a same-sized cell with respect to the parent cell, it must host virtual cells to contain a neighbor relation. In case it does, we transform the current virtual index across the respective entity to obtain the virtual cell index of the neighbor cell. Otherwise, we discard the neighbor relation. We also discard double-sized neighbors w.r.t. the parent cell, as the host cell interacts with one of the double-sized neighbor's virtual subcells. For

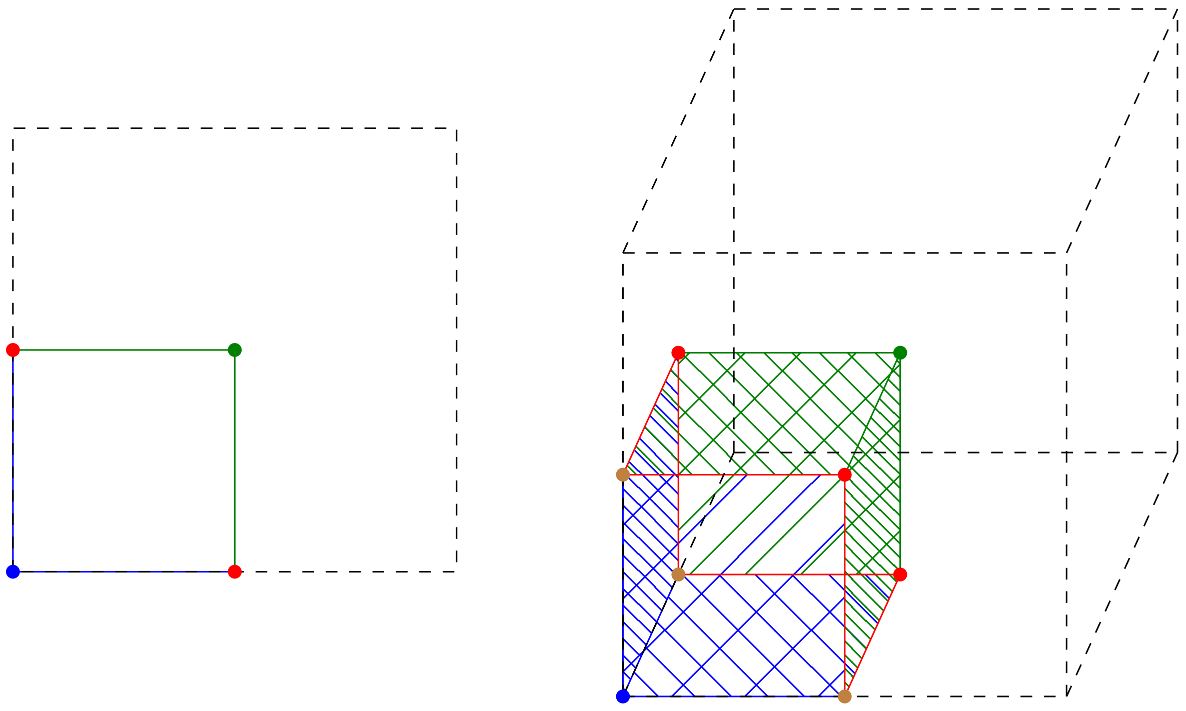


FIGURE 5.6 Illustration of different cases in neighbor search for an exemplary virtual cell. Green entities find their neighbors within the parent cell. Blue entities retain their search direction. For red entities, we change the search direction for the neighbor search originating from the host cell to a face, and for brown entities, we change the search direction to an edge. According to our definition from Sec. 2.1.1, red entities are face-hanging entities and brown entities are edge-hanging entities.

half-sized neighbors, we perform the same transformation that we used to find the virtual cell index of a real cell's neighbor to find the correct index in the list of hanging cells.

Summing up, in our case of a 2:1 balanced grid, we return the neighbors as stated in Tab. 5.1.

Facilitating Grid-Traversals Including Virtual Cells. For multi-variate time-stepping we have to traverse cells of a specific level. This can be done in two ways: Either we traverse the full grid and skip all cells with a different level than the currently active level, or we specifically only traverse those cells that have the desired refinement level.

We want to use the latter option and avoid touching all cells in each grid traversal. To encapsulate the complexity of finding the next cell when traversing different kinds of cells of a given level, we implement an iterator based on the information in `p4est_mesh` and `p4est_virtual` that exclusively traverses the cells of a given level. We support three criteria for selecting cells when creating the iterator, that is local or ghost cells, real cells or virtual cells as well as cells at the parallel boundary or inner cells. We can also traverse cells matching both criteria. Additionally, we integrate helper functions that facilitate

TABLE 5.1 Post-processing to return a same-sized neighbor for a neighbor search originating from a real or virtual cell given the neighbor or neighbors found using the neighbor search based on `p4est_mesh`.

	Same-sized neighbor	Double-sized neighbor	Half-sized neighbors
Real cell	Return neighbor	Find virtual cell	Discard neighbors
Virtual cell	Find virtual cell	Discard neighbor	Find correct cell

searching neighbors and calculating offsets for retrieving data.

This concludes our extensions to `p4est`. In the next chapter, we describe how to integrate our extended version of `p4est` into an existing application using the example of the ESPResSo simulation software.

5.2 Changing the Discretization in ESPResSo

The integration of our extended version of `p4est` with ESPResSo comprises several tasks according to our three-step integration procedure introduced in Sec. 1.3. First, we adapt the data-layout to the space-filling curve (SFC) and consider these changes in the implementations of numerical kernels. Then, we integrate these subcomponents which each other. To this end, we develop a common partitioning such that the data assigned to a given position in the simulation domain of all interaction partners are always located on the same process. We also develop a mapping between two `p4est` instances covering the same domain with a different discretization. Additionally, we create a model for dynamic grid adaptivity. Both previous points are closely related to dynamic load-balancing techniques.

5.2.1 Data-Layout and Adjusting Kernels

Changing the discretization from regular to dynamically-adaptive grids involves four crucial tasks. (i) We must change the data-layout such that it follows the SFC to make use of the locality imposed by the SFC for accessing data. (ii) This enforces a different grid-traversal where we no longer loop lexicographically over the data but linearly over a section of the SFC. (iii) This changes the way of accessing data of neighboring cells. (iv) For parallel simulations, we have to either change or extend the communication pattern such that it can fulfill the requirements of tree-structured grids. As the partitions are no longer cuboids and may consist of two separate parts when using certain curves, we can no longer rely on synchronous communication rounds.

During this thesis, we have ported the discretization of four logical components in ESPResSo from a regular Cartesian grid to using a dynamically-adaptive tree-structured grid based on p4est. We began by porting the implementation of the lattice-Boltzmann method (LBM) [1–3]. We use this implementation to verify that our extensions to p4est work beyond synthetic scenarios.

Then, Brunn ported the Linked-Cell method of ESPResSo's molecular dynamics (MD) implementation during his Master's thesis [4]. We introduce asynchronous communication and let p4est populate ESPResSo's internal data structures instead of exchanging kernels [205]. As already explained in Sec. 4.4, we abstain from extending the MD algorithm to adaptive grids.

Tischler ported the algorithms missing for solving the electrokinetic (EK) equations, that is the algorithm for the ionic flux and the electrostatic potential, during his Master's thesis [5]. We decided to use the same spatial discretization for the ionic flux, hydrodynamics, and the electrostatic potential.

For the grid-based algorithms (LBM, ionic flux, and electrostatic potential), we replace the lexicographically ordered data-structure by a per-level data in Morton-order including the payload of virtual cells. Additionally, we substitute loops over all coordinates with our iterator based on p4est_mesh (Sec. 5.1.2) and use p4est-based neighbor search instead of index calculations. Moreover, we replace ESPResSo's synchronous communication with the implementation we developed in p4est_virtual. Where possible, we adapt the kernels to match the new ways for accessing data. In case of the electrostatic potential, we implement the new algorithm (successive over-relaxation (SOR) instead of discrete Fourier transform (DFT), Sec. 4.3).

In the MD implementation, we do not touch the kernels, because the existing implementation is sufficiently generic to deal with non-cubic domain decompositions and cells ordered along a SFC. Here, it is sufficient to populate the algorithm's data structures using information from p4est and add a flag for asynchronous communication in the data exchange.

Summing up, we have two different p4est instances: One regular grid for the MD and one potentially adaptive grid for the three algorithmic components of the EK equations.

5.2.2 Coupling Physical Subsystems

To couple the different physical subsystems, we need a mapping between grids, i.e., we have to find cells that cover a given position $x \in \mathbb{R}^3$. Using the same discretization for ionic flux, hydrodynamics, and the electrostatic potential facilitates coupling significantly, because between these components partition boundaries, cell indices, and neighbors match by design. This leaves the coupling between particles and fluid.

ALGORITHM 5.1 Algorithm for finding a cell in a given p4est instance that contains a given position. We calculate the coordinate of a cell in the domain in tree coordinates (where each tree has a normalized side length of 1) and derive a virtual Morton index for this cell. We then binary search for this index in the other grid.

```
function encode_position($pos$)
  for all $p \in pos$ do                                   ⇝ Loop over given coordinates
    $c \leftarrow p * 2^{\ell_{\max}}$                     ⇝ Calculate lexicographic cell index
  end for
  return encode_morton($c$)                                ⇝ Calculate Morton-index for given cell index
end function

function pos_to_cell($p4est$, $pos$)                       ⇝ pos in p4est coordinates
  $pos\_idx \leftarrow$ encode_position($pos$)
  $cell\_idx \leftarrow$ binary_search(0, $n_{\text{cells}}$, $pos\_idx$)   ⇝ From cell's coordinate and level: derive overlap with virtual cell
  return $cell\_idx$
end function
```

Mapping between different discretizations. To couple between differently discretized p4est instances, we need a mapping between both discretizations. In other words, we have a cell containing a given position $x \in \mathbb{R}^3$ in p4est instance t_1, and we search the cell that contains that position in p4est instance t_2. To map between the discretizations of LBM and MD, both p4est instances have the same macrostructure, i.e., the different trees in the forest of octrees are arranged in the same way. This means, both instances use the same p4est_connectivity structure which describes the macrostructure of the grid. This implies that cells with the level ℓ have the same size in both grids. To generate a search space, we overlap the domain with a virtual regular single-tree grid that fully contains the entire simulation domain. We choose the mesh width such that finest level cells in the respective grid are equal to a single cell in the virtual regular grid. This restricts our simulation setup to "brick-like" scenarios where all octrees must have the same local orientation. We illustrate the idea in Fig. 5.7. We can now use Alg. 5.1 for mapping between different instances using the cell's coordinate. The virtual grid provides the virtual cell index of the position we want to localize in the second p4est instance. We search for this position in the second grid using binary search. This search yields the cell in the second instance covering the virtual cell's area which we are looking for.

For locating the blue dot in Fig. 5.7, we obtain virtual cell index 29 which we binary search in the adaptive grid. To this end, we calculate the virtual index of the front lower left corner of the cell. To correctly implement binary search with variable cell sizes in an adaptive grid, we must also consider the current cell's level, because coarser cells cover multiple virtual cells. We obtain the virtual index of the next cell by

$$i(c_{\text{next}}) = i(c_{\text{current}}) + 2^{\dim \cdot (\ell_{\max} - \ell(c_{\text{current}}))}. \tag{5.11}$$

This ensures that we detect hits correctly. If we, e.g., perform a binary search for virtual cell index 42 in the adaptive grid in Fig. 5.7, we have a grid of 17 cells and, thus, binary search starts at cell index 8. This cell has virtual cell index 32. Using Eq. (5.11), we detect that cell 8 is the correct cell.

Partitioning multiple `p4est` instances. We illustrate in Fig. 5.8 that we obtain a volume-to-volume communication problem if we have different domain decompositions for particles and fluid. This would require an additional binary search for each local particle and two rounds of communication: First, we have to find the process to which we have to send a varying number of particles, send these particles, couple them with the fluid, and transmit the result of the interaction. This issue arises either if we omit porting the MD implementation altogether or if we partition both grids independently. It makes large simulations infeasible.

Thus, we develop an algorithm to align partition boundaries and come up with a partitioning where the local cells of each process on both grids overlap perfectly. We achieve this by constructing the finest common tree (FCT) of both grids, Alg. 5.2. Note, that our algorithm requires aligned partition boundaries beforehand. We fulfill this initial condition by starting with two regular grids where we transfer the partition boundaries of the coarser grid to the finer grid. The algorithms presented in the following ensure that this constraint holds in the remaining simulation.

We coarsen all cells in `p4est` instance t_1, where cells in t_1 are finer than those in t_2 and vice versa. The algorithm traverses both `p4est` instances in parallel. One instance is automatically traversed using `p4est`'s coarsening function, and we follow in the second instance using a global state to save the cell index. We compare the level of the first cell of the family of cells that can technically be coarsened in instance t_1 (we traverse t_1 using `p4est`'s coarsening function) with the first overlapping cell in the second instance t_2 that we find. We coarsen, if the overlapping cell in t_2 is coarser than our current cell in t_1. As we do not constrain the level-difference between the instances t_1 and t_2 and create the FCT in a single iteration, we use recursive coarsening. In order to always find a matching cell for the first cell, we must catch those recursive coarsening calls. We calculate the virtual Morton index of the first cell and compare it to the one of the previous callback call. If the index is smaller, we subtract $2^{\dim} - 1$ from the iteration variable for the second instance.

After constructing the FCT, we partition it using variable cell weights as described in

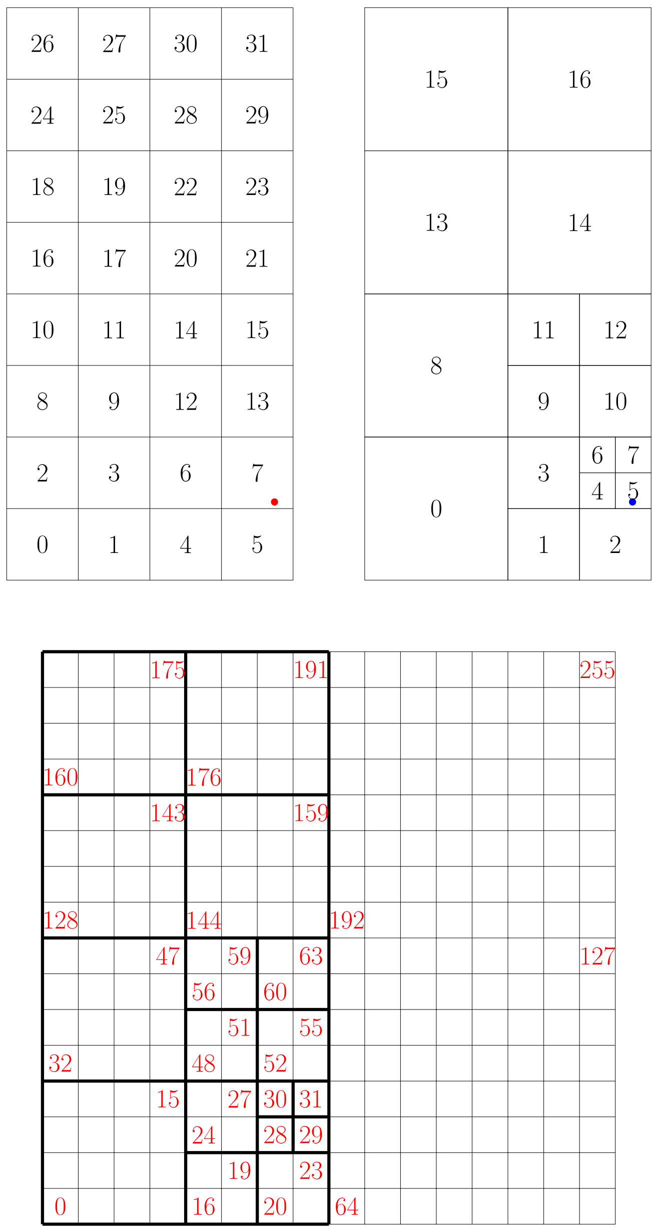


FIGURE 5.7 Illustration for calculating the cell index and retrieving the cell containing a given position (red or blue dot) for a potentially adaptive grid. We virtually overlap the domain with a regular Cartesian grid spanned by a single tree. This yields for maximally refined cells a single virtual cell index that matches the respective cell (cell 5 in the right grid) and an overlap region of $2^{\dim\cdot(\ell_{\max}-\ell)}$ virtual cell indices for coarser cells (e.g. cell 11 in the right grid). Thus, given we want to find the cell index of the red dot in the right grid, we search for the cell covering the virtual cell index 29 which is cell 5. For the regular grid on the left, our virtual grid has the refinement level of the regular grid. Virtual indices either match a cell index or are located outside the simulation domain. Thus, if we search in the opposite direction and look for the cell in the left grid that contains the blue dot, we search for the virtual cell index 7 which is cell 7.

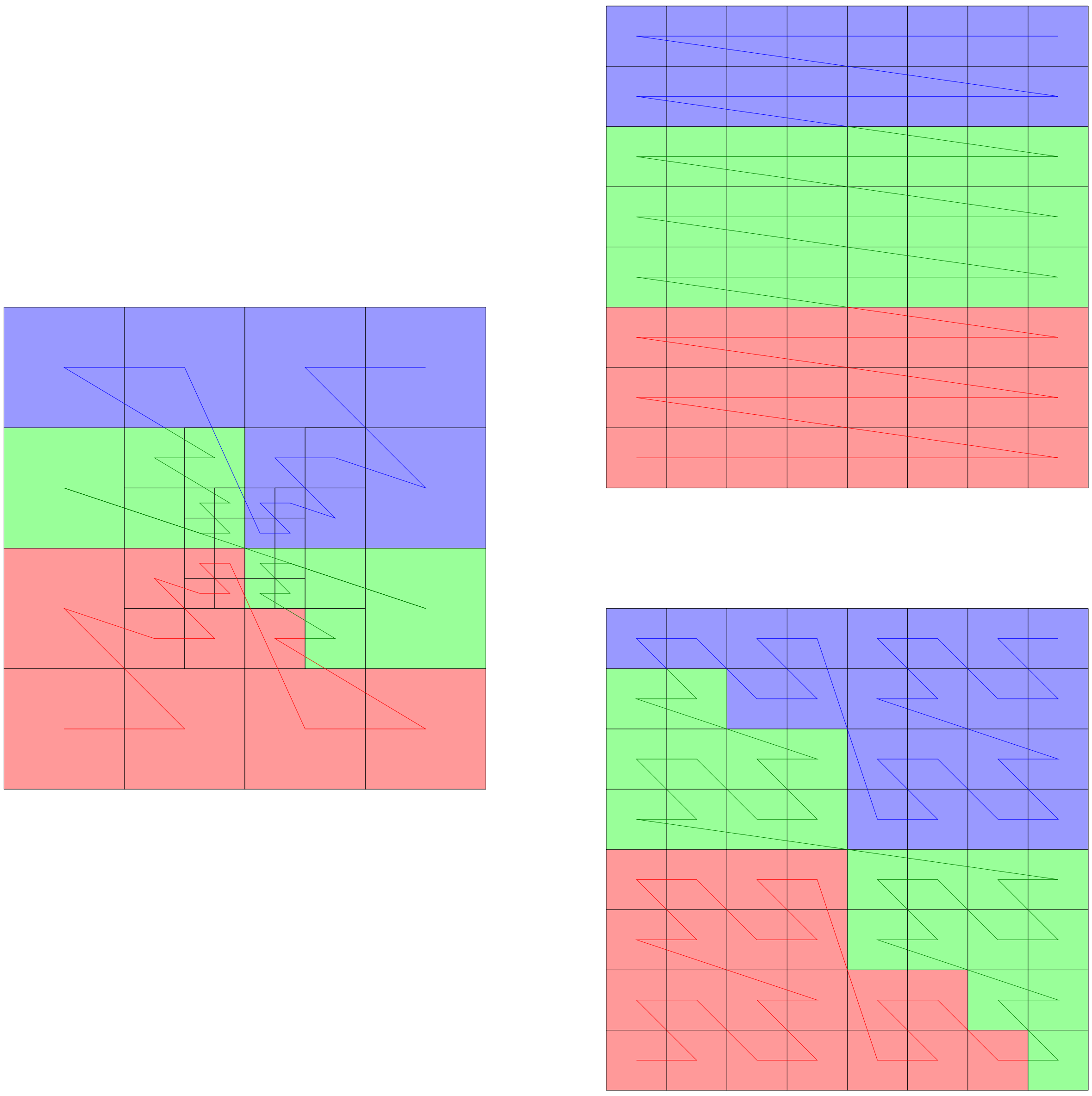


FIGURE 5.8 We compare different domain decompositions for coupling two systems with different discretizations. One grid is regular (right) and the other one is a tree-structured grid with a potentially arbitrary yet 2:1 balanced adaptive refinement pattern which may change between different integration steps (left). We see that we generally cannot match a regular Cartesian grid decomposition with lexicographic cell ordering with the SFC-based domain decomposition on the left (top-right). Moreover, we find that porting the regular grid to using a regular tree-structured grid and an SFC-based domain decomposition in itself is not a solution if the domain decompositions are created independently (bottom-right). In both cases, we obtain a volume-to-volume communication problem.

ALGORITHM 5.2 Algorithm to construct the FCT of two `p4est` instances t_1 and t_2 based on `p4est_coarsen` in `p4est` instance t_1. Within the coarsen-callback, we find a cell in t_2 that overlaps the first cell in the family of cells that may currently be coarsened using a depth-first postorder traversal scheme. We mark cells in t_1 for coarsening if their level is higher, i.e., they are smaller than their counterpart in t_2. As `p4est_coarsen` traverses one `p4est` instance, we use a global state to traverse the second instance concurrently. We detect recursive coarsening calls using virtual cell indices that we calculate using the function `virt_morton_idx` and saving the virtual index of the previous call. If we detect a recursive call, we modify the iteration variable for t_2 accordingly.

```
function create_fct($t_1$, $t_2$)                        ⇝ Generate finest common tree
  $fct \leftarrow$ p4est_copy($t_1$)
  $fct.user\_pointer \leftarrow t_2$                     ⇝ Let FCT contain both p4est instances
  $qid \leftarrow 0$                                     ⇝ Global variable, stores position in $t_2$
  $old\_morton\_idx \leftarrow 0$      ⇝ Global variable, stores virtual cell index of last callback call
  p4est_coarsen($fct$, coarsen_callback)                 ⇝ Initiate postorder traversal of grid
  return $fct$
end function

function coarsen_callback($t_1$, $tree\_no$, $qs$[])  ⇝ Recursively called for families of $2^{\mathrm{dim}}$ leaf cells
  $q \leftarrow qs$[0]
  $t_2 \leftarrow t_1.user\_pointer$                     ⇝ Extract second p4est instance
  if virt_morton_idx($q$) $< old\_morton\_idx$ then      ⇝ Handle recursive calls
    $qid \leftarrow \max\{0, qid - (\mathrm{P4EST\_CHILDREN} - 1)\}$
  end if
  $old\_morton\_idx \leftarrow$ virt_morton_idx($q$)
  repeat
    $p \leftarrow t_2(qid)$                              ⇝ Assign p from global variable
    $qid \leftarrow qid + 1$                             ⇝ Proceed in $t_2$
  until p4est_cell_overlaps($p$, $q$)
  return $p.level < q.level$                             ⇝ Coarsen cells if overlapping cell is coarser
end function
```

Sec. 2.1.3 [104, 208]. We calculate the partition-weight of a cell c_k in the FCT $w_i^{\mathrm{fct}}(c_k)$ as weighted sum over all input grids g,

$$w_i^{\mathrm{fct}}(c_k) = \sum_g \omega_g \sum_c w_{c,g}(c_j), \tag{5.12}$$

where $w_{c,g}(c_j)$ is the cell weight of all cells c_j in grid g that are contained in the current FCT-cell c_k and ω_g is the weight of the respective grid g. Then, we transfer the partition boundaries of the FCT to both instances.

We illustrate the full process of Alg. 5.3 in Fig. 5.9 for two input grids with an arbitrary yet aligned domain decomposition among three processes. The common partitioning

ALGORITHM 5.3 Algorithm for common partitioning. We calculate the FCT from both inputs grids, accumulate their weights for the FCT's cells, and partition the FCT. We convert this partitioning to the `p4est` instances t_1 and t_2, respectively. The inputs of this function besides t_1 and t_2 are weights for their cells returned by `repart_weights` and factors α_1 and α_2 for the linear combination of cell weights between trees.

```
function partition_jointly($\alpha_1$, $t_1$, $\alpha_2$, $t_2$)
  $fct \leftarrow$ create_fct($t_1$, $t_2$)
  for all $(s_f, s_1, s_2) \in fct.trees \times t_1.trees \times t_2.trees$ do        ⇝ Calculate weights for FCT
    for all $q \in s_f.quadrants$ do
      $Q_1 \leftarrow \{p \in s_1 : p$ overlaps $q\}$
      $Q_2 \leftarrow \{p \in s_2 : p$ overlaps $q\}$
      $w_1 \leftarrow \sum_{p \in Q_1}$ repart_weights($t_1$, $p$)
      $w_2 \leftarrow \sum_{p \in Q_2}$ repart_weights($t_1$, $p$)
      $W$[q] $\leftarrow \alpha_1 w_1 + \alpha_2 w_2$        ⇝ Combined repart weight for FCT cell
      $N_1$[q] $\leftarrow |Q_1|$        ⇝ Number of cells in $t_1$ for given FCT quad
      $N_2$[q] $\leftarrow |Q_2|$
    end for
  end for
  $\mathscr{P} \leftarrow$ determine_partitions($W$)
  for all $(r, P) \in \mathscr{P}$ do        ⇝ Calculate the number of cells per proc. $r$ for $t_1, t_2$
    $cells_1$[r] $\leftarrow \sum_{q \in P} N_1$[q]
    $cells_2$[r] $\leftarrow \sum_{q \in P} N_2$[q]
  end for
  p4est_partition_given($t_1$, $cells_1$)        ⇝ Transfer FCT partitioning to both grids
  p4est_partition_given($t_2$, $cells_2$)
end function
```

avoids additional volume-to-volume communication at the cost of load imbalances in both grids. In Tab. 5.2, we compare the effects of common partitioning to independent partitioning in terms of weight per process and the resulting imbalance in weights. We define local imbalance I_{local} as

$$I_{\text{local}} = \frac{\sum_{i=1}^{n_{\text{local cells}}} w(i)}{W} \tag{5.13}$$

the local sum of cell weights $w(i)$ over the global average of cell weights per partition W. Global imbalance I_{global} is defined as

$$I_{\text{global}} = \frac{\max_p \left(\sum_{i=1}^{n_{\text{local cells}}} w(i) \right)}{W} \tag{5.14}$$

TABLE 5.2 Examplary evaluation of load imbalances in local components induced by the common partitioning for the grids shown in Fig. 5.9. Given the uniform cell and grid weights as well as the resulting partitions from Fig. 5.9, we show the sum of local weights W as well as the local and global load imbalance. We show values for both grids and the FCT for independent and aligned partitioning. In the given scenario, our maximum load imbalance is bound by 1.2.

	Independent Partitioning				Aligned Partitioning					
Proc.	Adaptive		Regular		FCT		Adaptive		Regular	
	W	Imb.	W	Imb.	W	Imb.	W	Imb.	W	Imb.
Red	13	0.975	21	0.984	31	0.894	11	0.825	20	0.938
Green	13	0.975	21	0.984	35	1.010	16	1.200	19	0.891
Blue	14	1.050	22	1.031	38	1.096	13	0.975	25	1.172
Total	40	1.050	64	1.031	104	1.096	40	1.200	64	1.172

the maximum local sum of cell weights over the global average of cell weights per partition. Thus, imbalances below one indicate underloaded processes, imbalances above one indicate overloaded processes. For an imbalance value of one, the load is perfectly balanced.

Alg. 5.2 and Alg. 5.3 can be run in parallel with little communication for accumulating weights using global reduction and partial reduction for distributing cells to their new owners. `p4est` instances must initially have aligned partition boundaries. Thus, we align partitions once during startup when the grids of MD and LBM are both regular. We partition the finer grid according to the coarser one and ensure that their partition boundaries do not diverge from this point in time [207] by calling Alg. 5.3 for any repartitioning operation afterwards, e.g., after dynamically altering the adaptive grid.

After partitioning, we have to prepare the next integration step. We have to transfer the payload to their respective new owners and update metadata, i.e. regenerate `p4est_ghost`, `p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`. To transfer payload, we compare the partition tables before and after repartitioning. In the same way we overlap communication and computation in the main time-stepping loop, we allow updating `p4est` metadata while migrating the payload via `p4est_transfer_fixed`.

5.2.3 Dynamic Adaptivity in ESPResSo

We integrate the previously described partitioning scheme into our implementation of dynamic grid adaptivity. Our model is based on [209]. Implementing dynamic grid adaptivity, our central focus is on minimizing the memory-footprint, i.e., allocating new structures at the latest and freeing old structures as soon as possible.

To adapt the grid, we use multiple grid traversals and restrict ourselves to non-

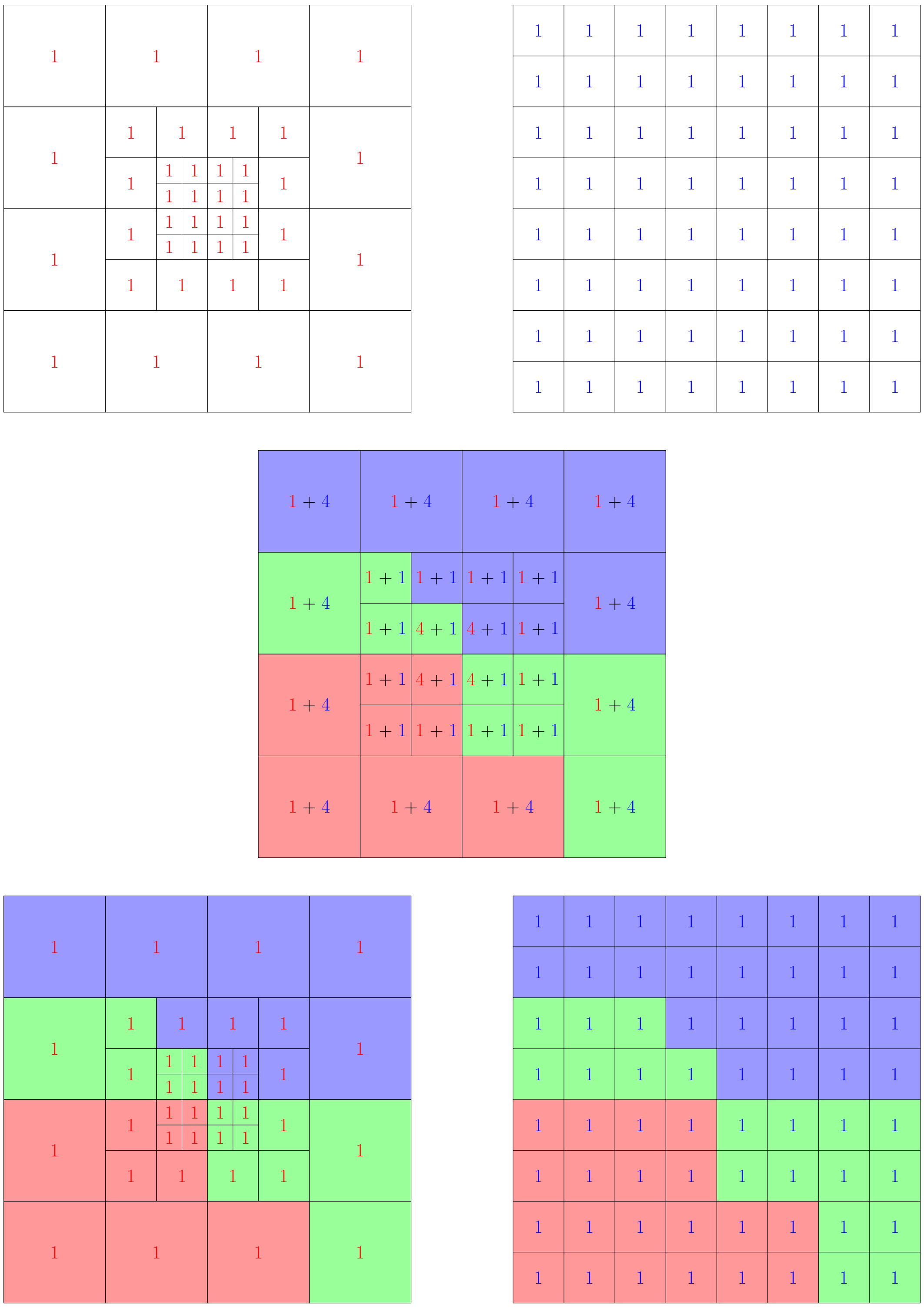


FIGURE 5.9 Example for the generation of a common partitioning based on the FCT of two grids. Top: Input grids with uniform cell weights and different discretization. Center: FCT of both grids that we partition according to its accumulated cell weight among the processes. Bottom: Transfer domain decomposition from FCT to both input grids.

ALGORITHM 5.4 Map data between grids for non-recursive grid-change, [210]. We refer to the data of a cell c as $data(c)$. We transfer the p4est instances before and after adapting the grid in parallel. By comparing the respective refinement levels we know which flag was set, and we can map the data accordingly.

```
function map_data($p4est\_old$, $p4est\_adapted$, $local\_data$, $mapped\_data$)
  $qid \leftarrow 0$
  for all $c \in local\_cells(p4est\_adapted)$ do
    $c\_ref \leftarrow p4est\_old(qid)$
    if $c.level \equiv c\_ref.level$ then                    ⇝ Cell was not changed
      $mapped\_data(c) \leftarrow local\_data$
      $qid \leftarrow qid + 1$
    else if $c.level + 1 \equiv c\_ref.level$ then            ⇝ Cell was coarsened
      interpolate($local\_data$, $qid$, $mapped\_data(c)$)
      $qid \leftarrow qid + 2^{\mathrm{dim}}$
    else if $c.level - 1 \equiv c\_ref.level$ then            ⇝ Cell was refined
      restrict($local\_data$, $qid$, $mapped\_data(c)$)
      $qid \leftarrow qid + 1$
    else
      raise(SIGABRT)                                     ⇝ Abort, must not happen
    end if
  end for
end function
```

recursive adaptivity. This means that cells may be either refined by one level, kept, or coarsened by one level. In our implementation, we prioritize refine over keep over coarsen. To take the decision, arbitrary cell-local functions might be implemented. We provide several pre-defined functions, based on information in different components. We allow refinement around obstacle cells and in (potentially moving) cuboids. Additionally, we refine the region around particles, i.e., cells taking part in the coupling between particles and fluid. Moreover, we provide several physical criteria. We can set threshold values for the fluid velocity $\boldsymbol{u}$, where we use the Euclidean norm, and the fluid's vorticity ζ, where we use the Chebyshev distance or maximum metric. In addition, there are threshold values for the gradient of densities ∇c and the gradient of the electrostatic potential $\nabla\Phi$.

Before actually changing the grid, we copy the p4est instance. Then, we locally refine, coarsen, and balance the grid on each process. By traversing both p4est instances, the original p4est and the adapted, simultaneously, we can map data from the old to the modified grid, see Alg. 5.4. This algorithm was introduced in the mangll-software, [210]. By design, there are three cases to consider for each cell, because it was either refined, kept, or coarsened. We choose to store the mapped data in a plain-array, because it facilitates data-transfer after repartitioning. After mapping data, we can free the data-structure of the local payload as well as all metadata for the old discretization.

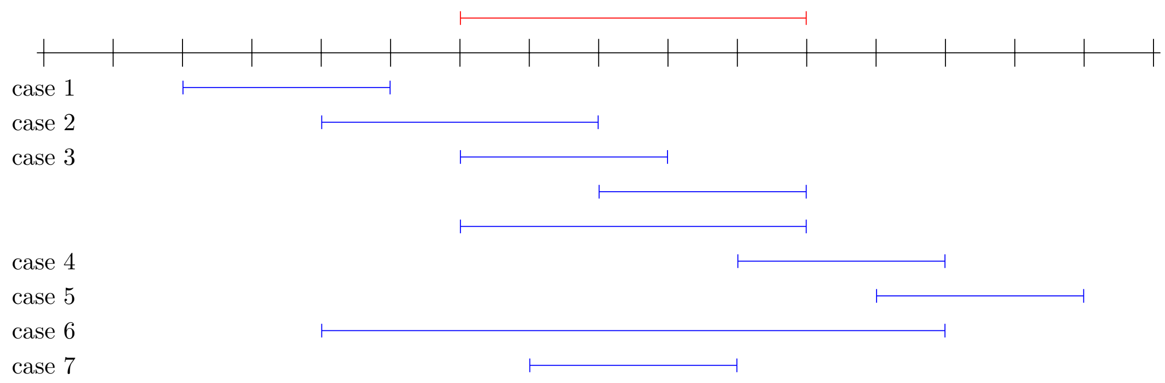


FIGURE 5.10 Cases for payload overlap before and after repartitioning. The process originally holding the red chunk along the SFC, may be assigned any of the blue chunks. Depending on which chunk it obtains, it has to exchange a certain number of cells with different processes.

We repartition the grid among processors according to Alg. 5.3 and transfer the payload accordingly to its new owner. To this end, we partition a copy of the adapted `p4est` instance. Thus, we keep the original partition table for sending and receiving data afterwards. Several cases may occur regarding the part of the SFC assigned to the current process before and after partitioning. (i) Former payload fully precedes new payload in SFC ordering. (ii) Former payload precedes new payload partially with partial or complete overlap over former payload. (iii) Former payload starts or ends at the same point as remote data with partial or full overlap. (iv) Former payload succeeds remote data with partial overlap. (v) Former payload succeeds remote data without any overlap. (vi) Former payload is a subset of the new payload. (vii) Former payload is a superset of the new payload. We illustrate these cases in Fig. 5.10.

Given the old and the new partition table, we can calculate which cells to send from process i to process j using Eq. (5.15) and which cells to receive on process i from process j using Eq. (5.16) [85, 211].

$$n^i_{\text{cells send, j}} = \max(0, \min(c^i_{\text{max old}}, c^j_{\text{max new}}) - \max(c^i_{\text{min old}}, c^j_{\text{min new}})) \tag{5.15}$$

$$n^i_{\text{cells recv, j}} = \max(0, \min(c^j_{\text{max old}}, c^i_{\text{max new}}) - \max(c^j_{\text{min old}}, c^i_{\text{min new}})) \tag{5.16}$$

We denote the minimum and maximum cell index on rank k by c^k_{min} and c^k_{max}. We transfer data from a plain array in Morton-ordering, which we directly use as send-buffer, into a dedicated receive buffer. By design, the data within the receive-buffer are in Morton-ordering and all local cell data that remains on the respective rank are moved to the respective index within the receive-buffer.

We free the adapted yet not partitioned p4est instance and initiate the actual data-transfer. While the processes exchange messages, we create new instances of p4est_ghost, p4est_mesh, p4est_virtual, and p4est_virtual_ghost. As building p4est_ghost also involves communication, we must tag messages accordingly. Generally, tagging messages is optional, however, it becomes mandatory to distinguish messages if we exchange multiple messages of the same data-type between the same sender and receiver. We wait for the data-transfer to complete and free our send-buffers. Before the next integration step can begin, we re-create per-level data-structures by traversing the receive-buffer and move the data to the respective position in the per-level array. Populating the data of ghost-replicas completes Alg. 5.5.

5.3 Summary

In this chapter we have illustrated all necessary steps for integrating p4est with ESPResSo. We have extended p4est to facilitate random-access and added support for virtual cells. Here, we emphasize that we have not touched existing internal algorithms and, thus, not impaired their scaling behavior. In ESPResSo, we have ported four physical subsystems to a p4est-based discretization. To ensure local coupling, we present an algorithm to align partition boundaries of multiple distinct p4est instances having the same macrostructure using the finest common tree (FCT). Moreover, we have developed a concept for dynamic spatial adaptivity and integrated it with a dynamic load balancing mechanism. In the following chapter, we prove that our implementations are physically correct and analyze their scaling behavior.

ALGORITHM 5.5 Algorithm to dynamically adapt the grid at run time. We collect refinement markers using arbitrary user-defined criteria per cell, where `refine` is stronger than `keep` which is stronger than `coarsen`. Then, we adapt a copy of the grid according to the markers previously collected and re-establish 2:1 balancing. We traverse the original grid and the adapted grid in parallel and map the numerical payload to the new discretization. Here, we use the plain linear storage scheme. Now, we can discard all remaining metadata and partition a copy of the adapted grid. We compare the partition tables to create MPI requests for sending the payload. These are processed while we create updated metadata for the grid. Finally, we store data in the per-level data structure.

```
function adapt_grid
    ghost_data ← ⊥                                             ⇝ Discard ghost data
    p4est_copy ← p4est               ⇝ Save original discretization for transferring data
    collect_markers(flags)
    p4est_refine(p4est_copy)
    p4est_coarsen(p4est_copy)
    p4est_balance(p4est_copy)
    map_data(p4est, p4est_copy, local_data, mapped_data)   ⇝ Map data to new discretization, Alg. 5.4
    p4est_ghost, p4est_mesh, p4est_virtual, p4est_virtual_ghost, local_data ← ⊥   ⇝ Reset level-wise payload, and metadata
    p4est ← p4est_copy                                         ⇝ Backup old partition table
    partition_jointly(1.0, p4est_copy, 1.0, p4est_md)   ⇝ Repartition, Alg. 5.3, fixed uniform grid weights
    status ← p4est_transfer_fixed_begin(p4est, p4est_copy, mapped_data, recv_buffer)   ⇝ Initiate transfer of payload
    p4est ← p4est_copy
    rebuild_p4est_meta_data(p4est)             ⇝ Rebuild p4est_ghost, p4est_mesh, ...
    p4est_transfer_fixed_end(status)                           ⇝ End pending data transfer
    unflatten_data(local_data, recv_buffer)              ⇝ Restore per-level data-structure
    p4est_ghost_exchange_data(local_data, recv_buffer)   ⇝ Replicate payload to neighboring processes
end function
```

6 Computational Results

In this chapter we test our implementation. First, we verify its physical correctness by simulating scenarios with well-known behavior and analytical solution. Then, we investigate the run time performance and scaling behavior of the lattice-Boltzmann method (LBM) and the short-range molecular dynamics (MD) implementation as well as of the coupling between them. Finally, we consider a nanopore setup similar to [6, 26].

6.1 Testing the Implementation

We test the physical correctness of our implementation using publicly available ESPResSo system test cases[1] and add a simple molecular dynamics (MD) test for energy conservation. Couette and a Poiseuille flow serve as test cases for our lattice-Boltzmann method (LBM) implementation, a microcanonical ensemble (NVE ensemble with constant number of particles, constant volume, and constant energy) to verify energy conservation of our MD implementation, and an electro-osmotic flow for the continuous electrokinetic model. Similar tests using this implementation were already performed in [1, 2] for the LBM, in [4, 205] for MD, and in [5] for the full electrokinetic system.

6.1.1 Lattice-Boltzmann Method

We verify our LBM implementation by simulating a Couette flow and a Poiseuille flow. For these simple scenarios, the analytical solution is well-known and understood. We choose a cubical domain with side length 8 with periodic boundaries in x and y directions and walls positioned at $z = 0.5$ and $z = 7.5$, see Fig. 6.1. We discretize the domain by a single octree.

[1]`https://github.com/espressomd/espresso/tree/python/testsuite/python`

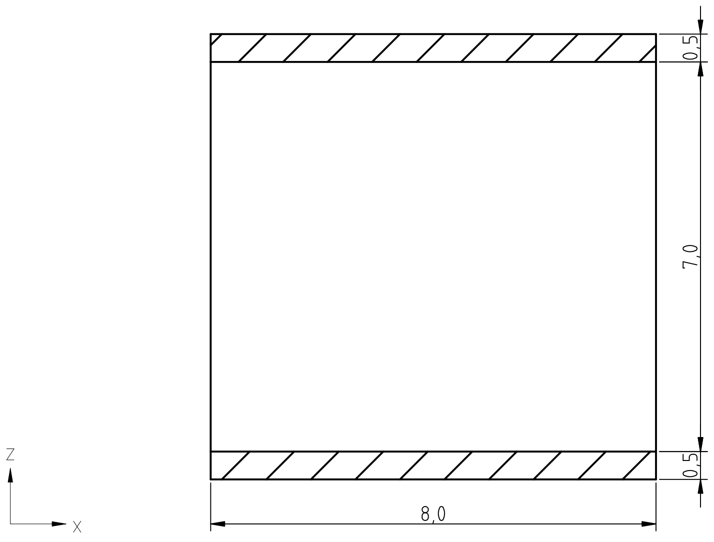


FIGURE 6.1 Verification scenario for our implementation of the LBM on adaptive grids. We show a cut through the 3d geometry in the x-z-plane. Walls enclose the domain in z direction, in x and y directions we use periodic boundaries.

In a Couette flow, we simulate a flow between two plates with no-slip boundary conditions. The lower plate is fixed and the upper one is moving with a constant velocity. At the upper plate, we set a velocity Dirichlet boundary condition with $\tilde{\boldsymbol{u}} = (0.1, 0.2, 0.0)^T$. We expect a linear velocity profile between both plates in x and y direction as well as for the total velocity. We use two regular grids of level six and seven as well as two randomly refined, statically adaptive grids. To generate the latter, we use regular grids of level four and five, refine them twice at random locations, and re-establish 2:1 balancing afterwards. For all grids, we see good agreement with the analytical solution, see Fig. 6.2 and Fig. 6.3.

Poiseuille flow is a flow between two static plates with no-slip boundary conditions. Here, we drive the flow by velocity boundary conditions. We enlarge the domain by a factor of four in x-direction to $(32, 8, 8)$ and add walls acting as velocity Dirichlet boundary conditions at $x = 0.5$ and $x = 31.5$ of $\tilde{\boldsymbol{u}} = (0.1, 0.0, 0.0)^T$. Thus, we discretize the domain by concatenating four octrees, Fig. 6.4

We derive the analytical solution from the volumetric flow rate Q of a Poiseuille flow for a given channel surface A, which is given as

$$Q = \bar{\boldsymbol{u}}A = \frac{2}{3}\boldsymbol{u}_{\max}A \tag{6.1}$$

where $\bar{\boldsymbol{u}}$ and $\boldsymbol{u}_{\max}$ denote the average and maximum fluid velocity [56]. Eq. (6.1) allows fitting a parabola through the maximum velocity value $\boldsymbol{u}_{\max} = \frac{3}{2}\bar{\boldsymbol{u}} = \frac{3}{2}\tilde{\boldsymbol{u}}$ at $z = 4$ and both boundaries where the velocity is 0.

Using the same discretizations as in the Couette flow scenario, we again see good

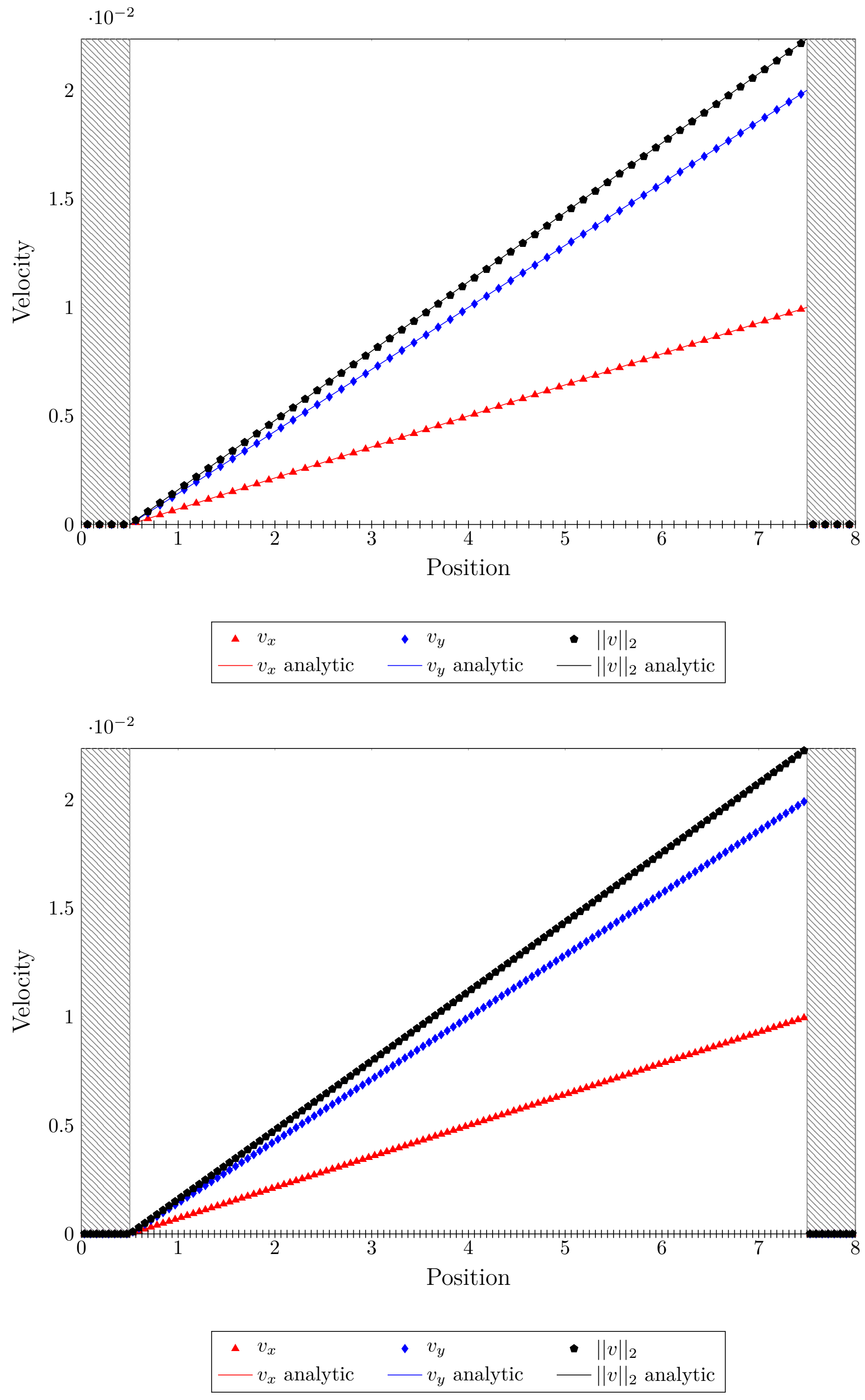
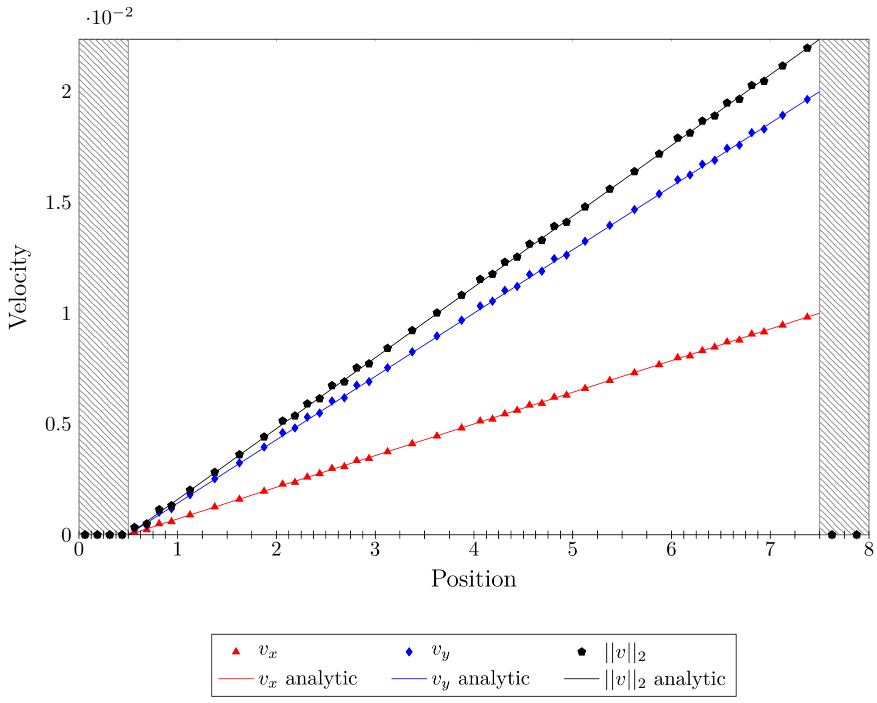


FIGURE 6.2 Validation test case for the LBM solver. We show the flow profile of a Couette flow for the regular grids of level six (top) and level seven (bottom) with a velocity Dirichlet boundary condition of $(0.1, 0.2, 0.0)^T$ plotted from $(4.0, 4.0, 0.0)$ to $(4.0, 4.0, 8.0)$. Hatched areas indicate walls. Ticks along the bottom axis illustrate the discretization.



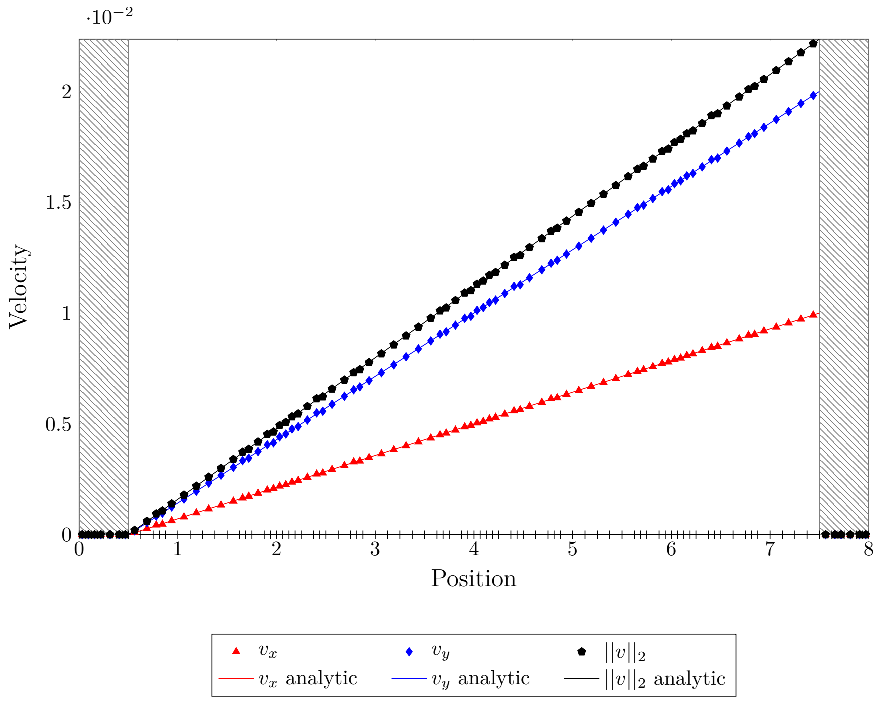


FIGURE 6.3 Validation test case for the LBM solver. We show the flow profile of a Couette flow for statically adaptive grids with a velocity Dirichlet boundary condition of $(0.1, 0.2, 0.0)^T$ plotted from $(4.0, 4.0, 0.0)$ to $(4.0, 4.0, 8.0)$. Hatched areas indicate walls. Statically adaptive grid randomly refined twice from level four (top) and level five (bottom). We establish 2:1 balancing once after refinement. Ticks along the bottom axis illustrate the discretization.

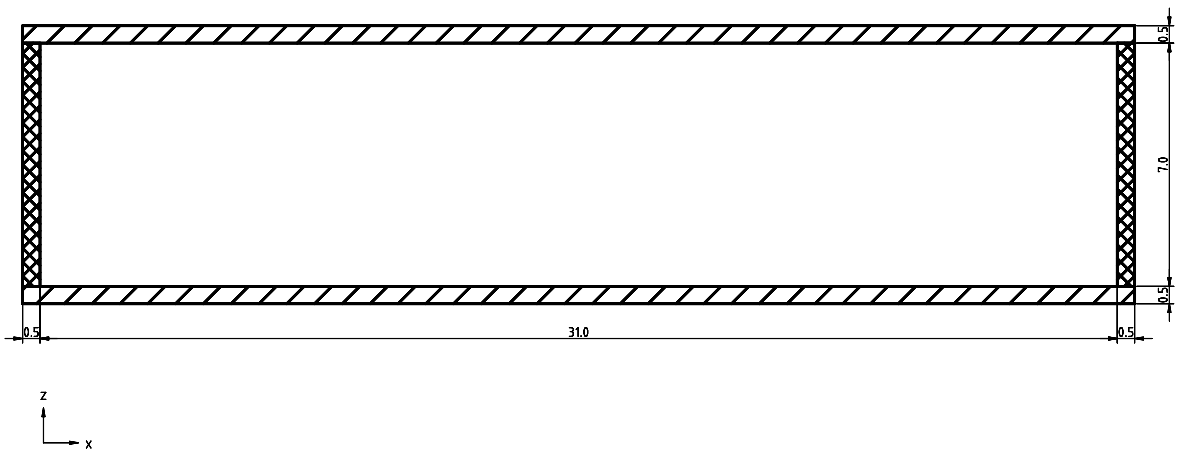


FIGURE 6.4 Prolonged verification scenario for the velocity-driven Poiseuille flow on adaptive grids. We show a cut through the 3d geometry in the x-z-plane. Walls (hatched areas) enclose the domain in x and z direction, in y direction we use periodic boundaries. Crosshatched walls serve as velocity Dirichlet boundary conditions.

agreement with the analytical solution, see Fig. 6.5 and Fig. 6.6.

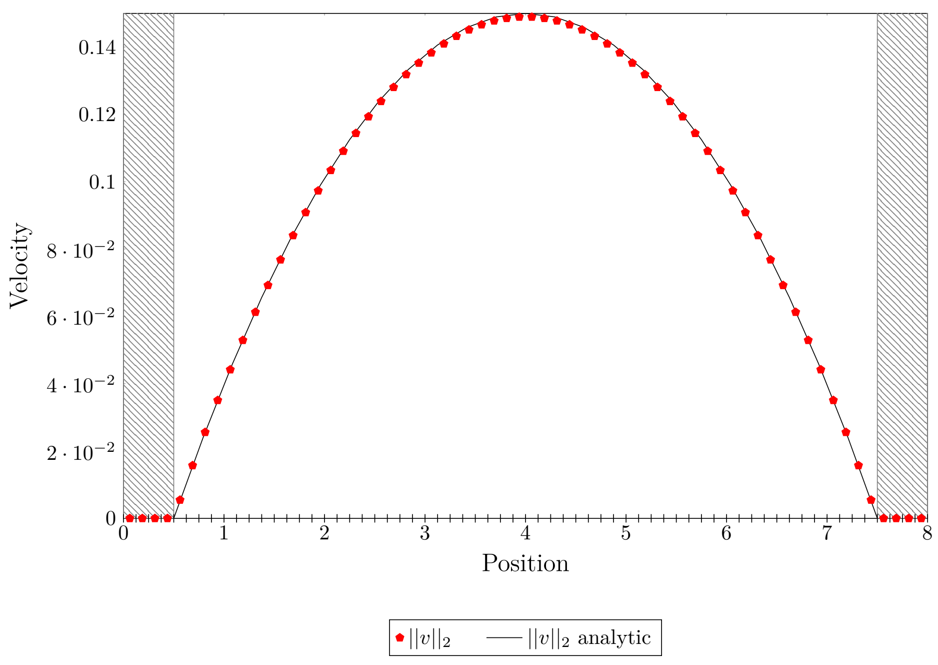
6.1.2 Molecular Dynamics

To prove the correctness of our short-range MD implementation based on regular `p4est` grids as a Linked-Cell structure, we consider the energy of a microcanonical ensemble of 600 particles. We model the short-range interaction by a Lennard-Jones (LJ) potential with a cut-off radius $r_c = 2.5$, $\varepsilon = 1$, and $\sigma = 1$. Initially, we distribute the particles randomly within a cubical domain with a side length of 80 and assign in each dimension a random velocity between 0 and 0.1. This yields a total velocity between 0 and $0.1 \cdot \sqrt{3}$. We obtain a regular Linked-Cell grid consisting of a single tree with a refinement level of 5.

We equilibrate the system and analyze the energy of the system for 10,000 time steps. Fig. 6.7 shows that energy is conserved as expected.

6.1.3 Electrokinetics

We verify the electrokinetics implementation using an electro-osmotic flow. In this scenario, an ionic fluid and a charged surface are subjected to an external electrical field. The ionic fluid forms a double layer where positive ions accumulate close to a negatively charged surface. The surface attracts positively charged ions and repulses negatively charged ions. From the electric field and this distribution of ions, a net motion of ions in the direction of the electric field is induced. This leads to a flow in the direction of the electric field [29]. While complex phenomena may occur in complex geometries [212],



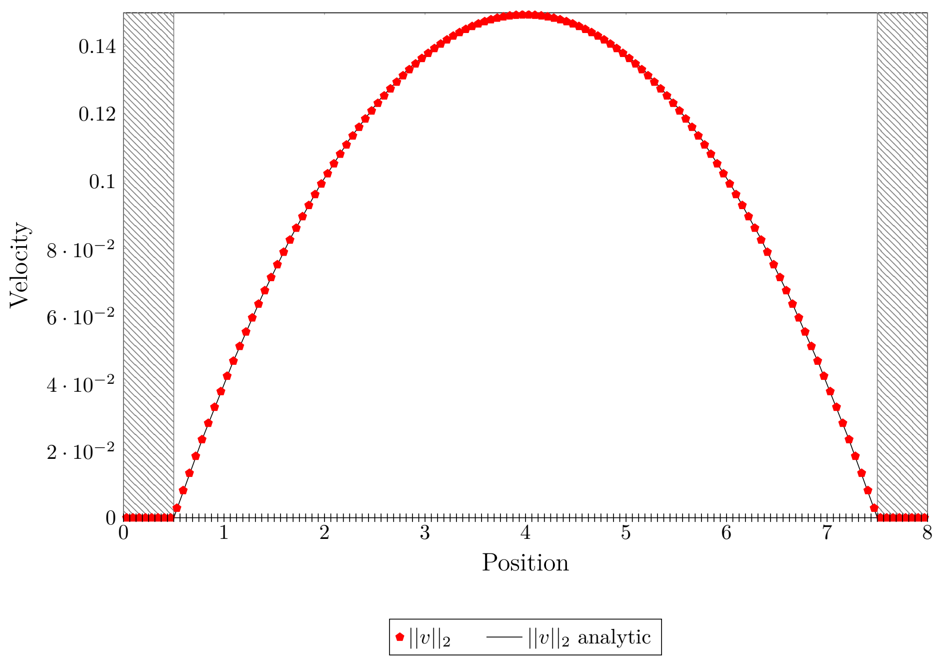


FIGURE 6.5 Validation test case for the LBM solver. We show the flow profile of a Poiseuille flow for regular grids of level six (top) and seven (bottom) with a velocity Dirichlet boundary condition of $(0.1, 0.0, 0.0)^T$ plotted from $(16.0, 4.0, 0.0)$ to $(16.0, 4.0, 8.0)$. Hatched areas indicate walls. Ticks along the bottom axis illustrate the discretization.

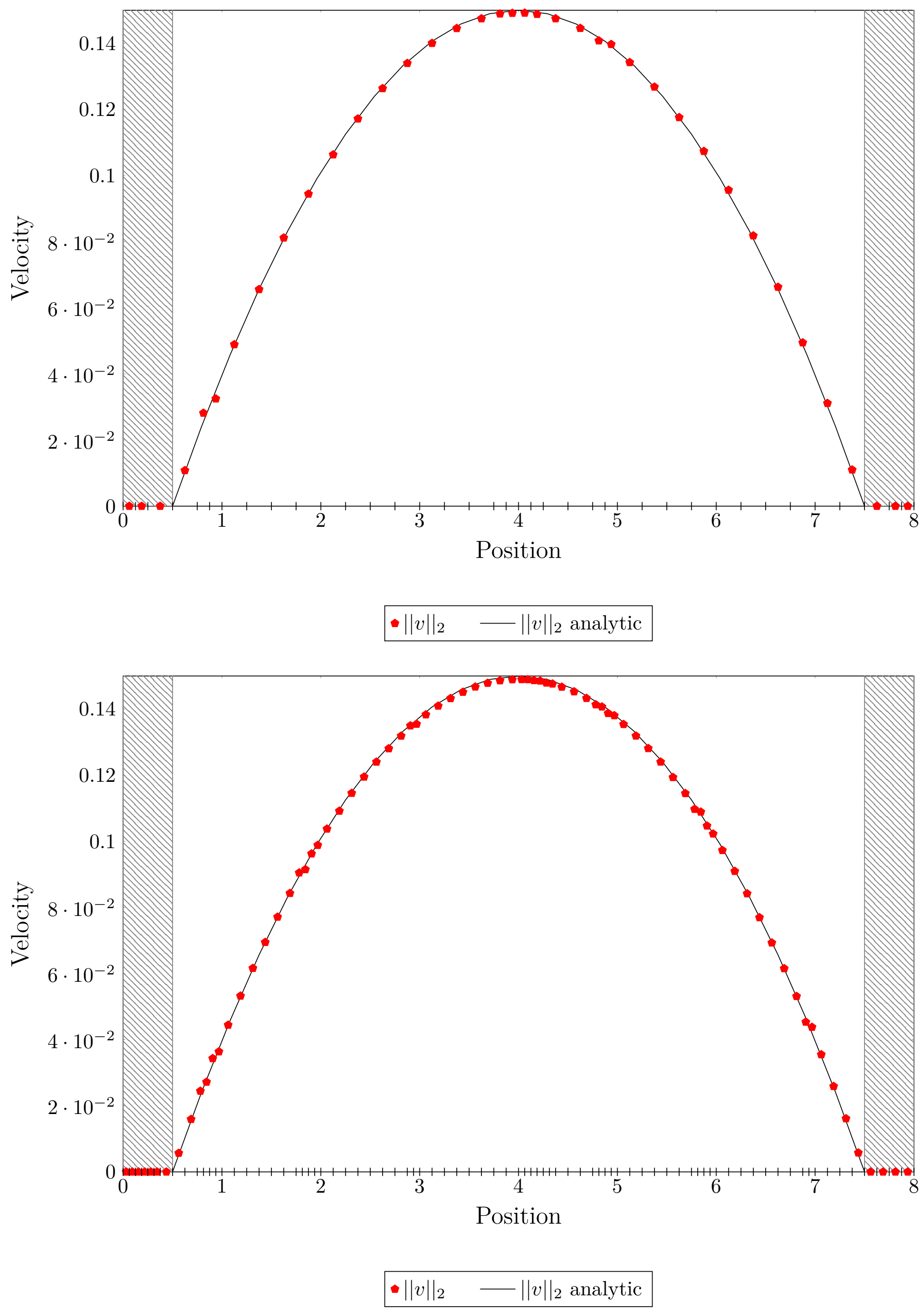


FIGURE 6.6 Validation test case for the LBM solver. We show the flow profile of a Poiseuille flow for statically adaptive grids with a velocity Dirichlet boundary condition of $(0.1, 0.0, 0.0)^T$ plotted from $(16.0, 4.0, 0.0)$ to $(16.0, 4.0, 8.0)$. Hatched areas indicate walls. Top: statically adaptive grid randomly refined twice from level four. Bottom: statically adaptive grid randomly refined twice from level five. We establish 2:1 balancing once after refinement. Ticks along the bottom axis illustrate the discretization.

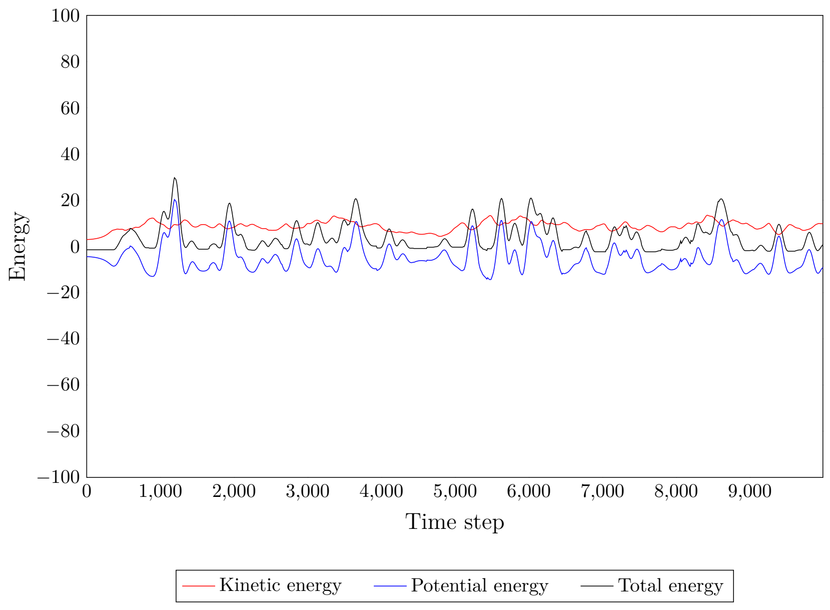


FIGURE 6.7 Energy of an equilibrated microcanonical ensemble of 600 particles over 10,000 time steps.

analytical solutions can be derived for simple geometries [213]. In case of a simplistic slit-pore geometry, where we have two infinite parallel plates, we can derive the analytical solution for both flow profile and ionic density normal to the plates. The solution for the ionic density profile $c(x)$ is given as

$$c(x) = \frac{\xi^2}{2 \cdot \pi \cdot \ell_B \cdot \cos^2(\xi \cdot x)} \tag{6.2}$$

where ℓ_B denotes the Bjerrum length and ξ is a constant based on the geometric and physical properties of the system. Relevant parameters for ξ are the distance between both plates d, the Bjerrum length ℓ_B, the valency z, and the surface charge density σ. We calculate ξ using bisection. To this end, we search the zero point of

$$g(\xi) = \xi \cdot \tan\left(\frac{\xi \cdot d}{2}\right) + 2 \cdot \pi \cdot \ell_B \cdot \frac{\sigma}{z} \tag{6.3}$$

at three points. Initially, we use $\{\xi_a = 0, \xi_b = k, \xi_c = 2k\}$ with step size $k = \frac{\pi}{2 \cdot d}$. Then, we search the zero value in the interval (ξ_a, ξ_c) using Alg. 6.1. The algorithm works if and only if there is exactly one solution for $g(x) = 0$ in the interval. Additionally, $g(\tilde{x})$ and $g(x')$ must have opposite sign with $\xi_a < \tilde{x} < x$ and $x < x' < \xi_c$. For the system at hand, we have $\xi \simeq 0.0515$.

ALGORITHM 6.1 Bisection scheme for evaluating the system constant ξ. We search for the zero value of g in the interval $(0, \frac{\pi}{2\cdot d})$. The algorithm works if and only if there is exactly one solution for $g(x) = 0$. Additionally, $g(\tilde{x})$ and $g(x')$ must have opposite sign with $0 < \tilde{x} < x$ and $x < x' < \frac{\pi}{2\cdot d}$.

```
function bisect($min\_step, d$)
    $step \leftarrow \frac{\pi}{2\cdot d}$                                    ⇝ d: distance between plates
    $pa \leftarrow 0$
    $pb \leftarrow step$
    $pc \leftarrow 2 \cdot step$
    while $step > min\_step$ do
        $vala \leftarrow g(pa)$                           ⇝ Evaluate Eq. (6.3) at pa, pb, and pc
        $valb \leftarrow g(pb)$
        $valc \leftarrow g(pc)$
        $step \leftarrow 0.5 \cdot step$                                          ⇝ Half step size
        if (($vala < 0$ and $valb < 0$ and $valc > 0$) or ($vala > 0$ and $valb > 0$ and $valc < 0$)) then    ⇝ Zero between [pb, pc]
            $pa \leftarrow pb$
            $pb \leftarrow pb + step$
        else if (($vala > 0$ and $valb < 0$ and $valc < 0$) or ($vala < 0$ and $valb > 0$ and $valc > 0$)) then    ⇝ Zero between [pa, pb]
            $pc \leftarrow pb$
            $pb \leftarrow pb - step$
        else
            raise(SIGABRT)                                ⇝ Abort, input condition not met
        end if
    end while
end function
```

The solution for the fluid profile $\boldsymbol{u}(x)$ is given as

$$\boldsymbol{u}(x) = \boldsymbol{f} \cdot \log\left(\frac{\cos(\xi \cdot x)}{\cos\left(\frac{\xi \cdot d}{2}\right)}\right) / (2 \cdot \pi \cdot \ell_B \cdot \eta \cdot \rho) \tag{6.4}$$

where ξ is the same system constant as for the ionic density in Eq. (6.2), $\boldsymbol{f}$ denotes an external force, η the fluid's dynamic viscosity, and ρ the fluid's density.

For our system we use the geometric setup shown in Fig. 6.8. We discretize the $4 \times 4 \times 60$ domain with 15 octrees, each with a side-length of four, and use a maximum refinement level of two.

We choose the physical parameters of the system as follows: We apply an external force $\boldsymbol{f} = (-0.13, 0, 0)^T$ and set the surface charge density $\sigma = 0.05$. The system contains two species, water as a neutral species with a density $\rho_{H_2O} = 26.15$ and counterions with a density of $\rho_c = 0.002$. We set the counterions' valency to $z = 1$ and the diffusion coefficient to $D = 0.3$. Moreover, we set the kinematic viscosity to $\eta = 2.3$, the friction

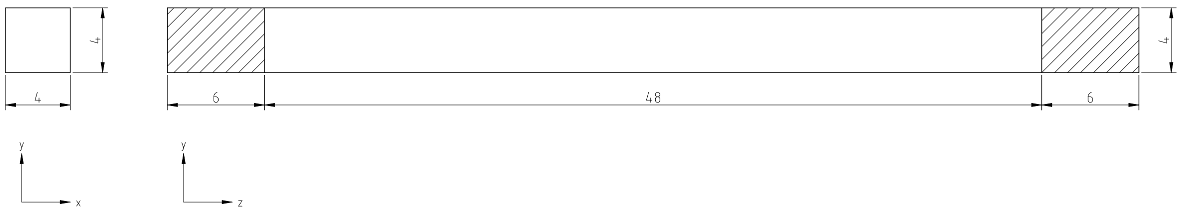


FIGURE 6.8 Verification scenario for the electrokinetic solver. We display the geometry for the electro-osmotic flow scenario. Hatched areas denote charged walls. We discretize the system with 15 octrees.

coefficient of the bidirectional force-coupling to $\Gamma = 4.3$, the Bjerrum length to $\ell_B = 0.47$, and use a constant global time step $\Delta t = \frac{1}{7}$.

We simulate the system until it is converged. To this end, we compare the velocity and the charge density in the domain's center, i.e., at $(2, 2, 30)$, every 256 fine grid time steps. We consider the system to have converged to its steady state if the accumulated absolute difference between both values is less than 10^{-7}.

We compare the density profile and the velocity profile of the converged simulation to the analytical solution for a regular grid of level two in Fig. 6.9. Results for the same scenario using an adaptive grid where we refine cells if the gradient of the density is larger than 0.0003 and coarsen cells if for each cell of a family the gradient is smaller than 0.0003 are visualized in Fig. 6.10.

Summing up, we have shown that we successfully ported the implementation of all four major physical components—hydrodynamics, ionic flux, electrostatic potential, and short-range MD—from a regular Cartesian grid to dynamically-adaptive tree-structured grids based on the forest-of-octree approach. We have changed the discretization while retaining the physical correctness of the implementation. In the following, we analyze the new implementation's run time performance and scalability.

6.2 Performance and Scalability

We test the performance and scalability of our implementation on the Tier 1 supercomputer "Hazel Hen" at the HLRS in Stuttgart, a Cray XC40 installation consisting of 7712 nodes, each of which is equipped with two twelve-core Intel Xeon E5-2680 processors and 128 GB of RAM. The processors are built in Intel's Haswell architecture. The nodes are connected by Cray's Aries Interconnect, and, at the time of our measurements, the system ran a Linux kernel in version 4.4.143. Tab. 6.1 gives more details about the machine's structure.

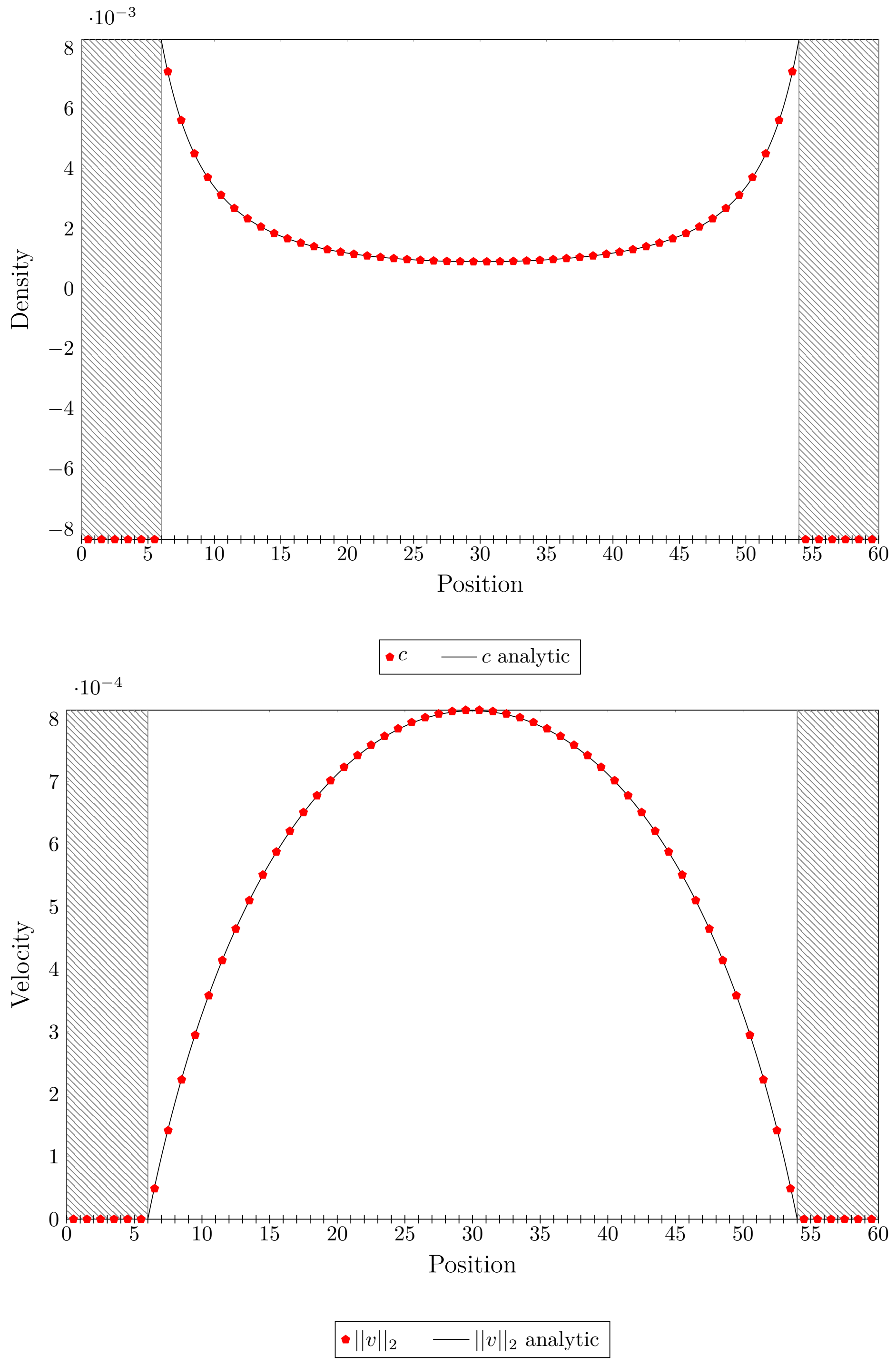
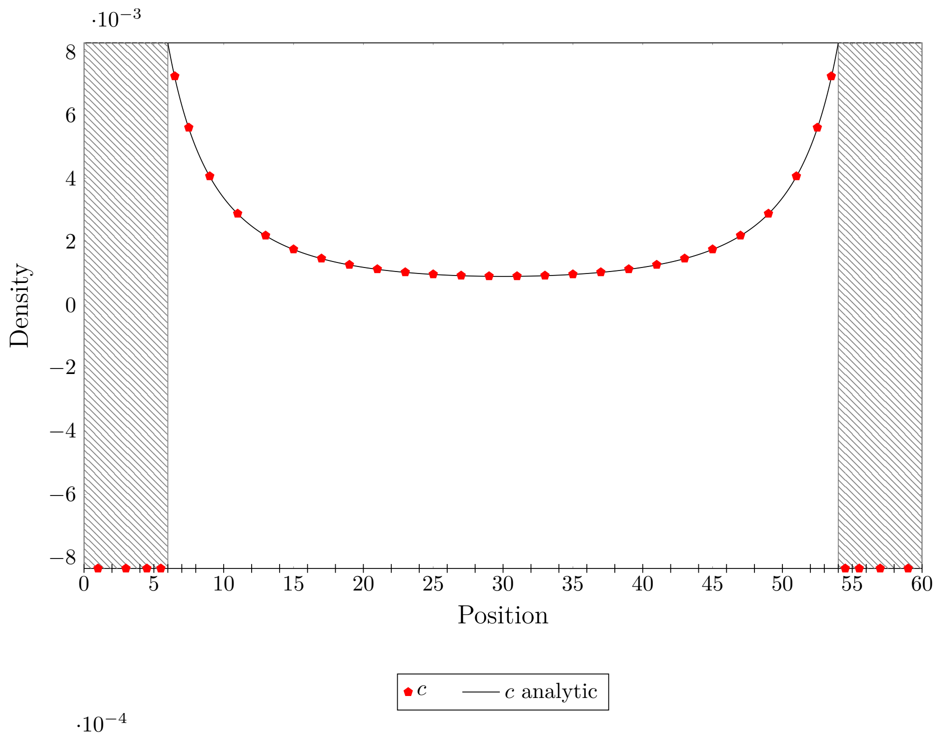


FIGURE 6.9 Validation test case for the electrokinetic solver. We show the comparison of the density profile (top) and the velocity profile of an electro-osmotic flow, discretized using 15 octrees with regular grids of level two, with the analytical solution. We plot the profile from $(2.0, 2.0, 0.0)$ to $(2.0, 2.0, 60.0)$. Hatched areas denote walls, ticks along the bottom axis illustrate the discretization.



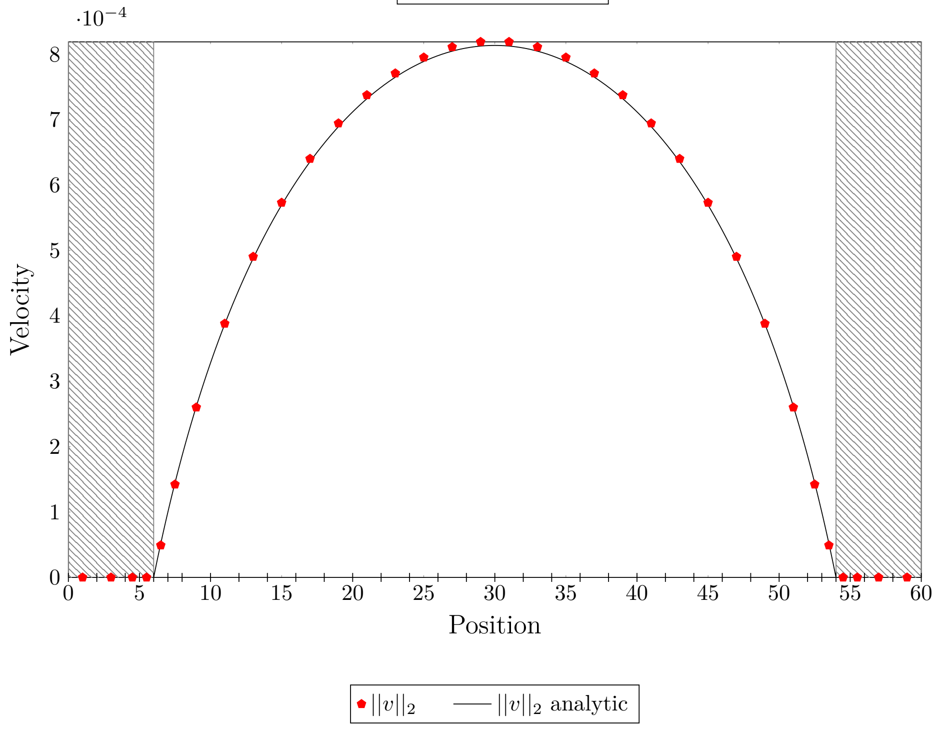


FIGURE 6.10 Validation test case for the electrokinetic solver. We show the comparison of the density profile (top) and the velocity profile of an electro-osmotic flow, discretized using an adaptive grid with maximum refinement level two, with the analytical solution. We use a threshold value of 0.0003. Whenever the gradient of the density exceeds the threshold, we refine the cell. We coarsen families of cells where for each cell the gradient of the density is below the threshold. We plot the profile from $(2.0, 2.0, 0.0)$ to $(2.0, 2.0, 60.0)$. Hatched areas denote walls, ticks along the bottom axis illustrate the discretization.

TABLE 6.1 Structure of the Hazel Hen supercomputer. We abbreviate Cabinet Groups by "Cab. Grps.".

	Cores	Processors	Nodes	Blades	Chassis	Cabinets	Cab. Grps.
Processors	12	1					
Nodes	24	2	1				
Blades	96	8	4	1			
Chassis	1536	128	64	16	1		
Cabinets	4608	384	192	48	3	1	
Cab. Grps.	9216	768	384	96	6	2	1
Machine	185,088	15,424	7712	1928	121	41	21

6.2.1 Lattice-Boltzmann Method

To investigate the scalability of our implementation of the lattice-Boltzmann method (LBM), we simulate a driven cavity in a cubical domain with a side length of 8. We discretize the domain by a single octree. The geometric setup is shown in Fig. 6.11, where hatched areas denote walls. On each wall, we apply no-slip boundary conditions. The top wall moves with a constant velocity of $(0.1, 0.0, 0.0)^T$, the other walls are static. In y-direction, we use periodic boundaries.

We investigate the scaling behavior for two regular grids, a statically refined grid (maximum refinement at geometric boundaries and in the cross-hatched region indicated in Fig. 6.11), and a dynamically adaptive grid where we move the refinement region in y-direction by one fine grid cell every two coarse time steps. We simulate 160 fine grid time steps, thus, in case of dynamic adaptivity, we adapt the grid ten times. We overlap communication with computation during the LBM step and while sending data after adapting the grid as explained in Sec. 5.2. We show the benefit in performance for all types of grids and for both scaling experiments in Fig. 6.12. We show the performance enhancement as $\frac{t_{\text{comm max, not hidden}} - t_{\text{comm max, hidden}}}{t_{\text{comm max, not hidden}}}$. While we see a consistent benefit for strong scaling, in weak scaling this is only the case for regular grids. For adaptive grids there are scenarios where we benefit and others where we do not. The massive break down for data transfer in weak scaling may be related to concurrently creating `p4est_ghost` and, thus, putting a lot of load on the network. The remaining occasions where this occurs are outliers resulting from comparing the maximum values.

To partition the load among processors using space-filling curve (SFC)-based partitioning, we assign each cell a uniform weight. Specifically, we do not take into account if a cell is a boundary cell or a fluid cell. This reduces load imbalance, e.g., for building metadata, specifically when building `p4est_mesh` and `p4est_virtual`, at the cost of introducing load imbalance during the LBM step. We decided to use a simple scheme,

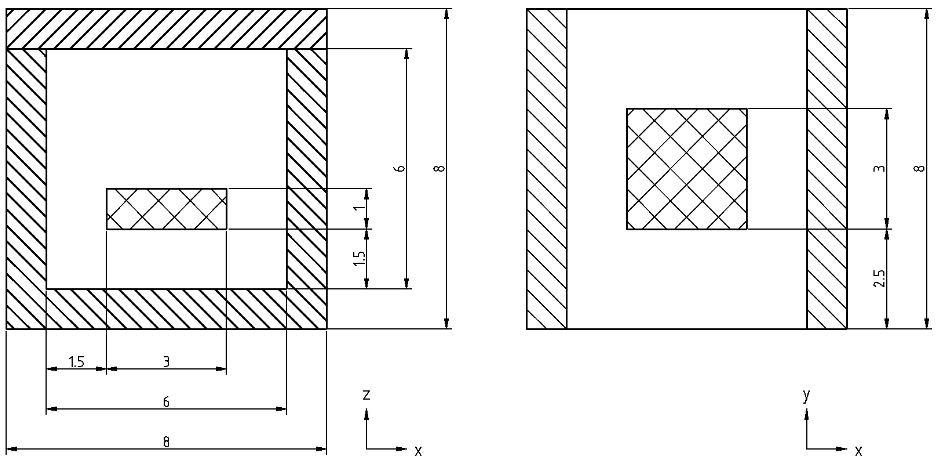


FIGURE 6.11 Geometric setup for LBM scaling experiments (2D projections of the 3D domain). Linearly hatched areas denote geometric boundaries, in y-direction we use periodic boundaries. Each wall has no-slip boundary conditions. We set a constant Dirichlet velocity boundary condition of $(0.1, 0, 0)^T$ at the top wall. The remaining walls are static. The crosshatched box within the simulation domain marks the area of maximum refinement within the simulation domain for adaptive simulations. We use a square base and, in case of dynamic adaptivity, we move this region by one fine cell after two coarse time steps along the y-axis.

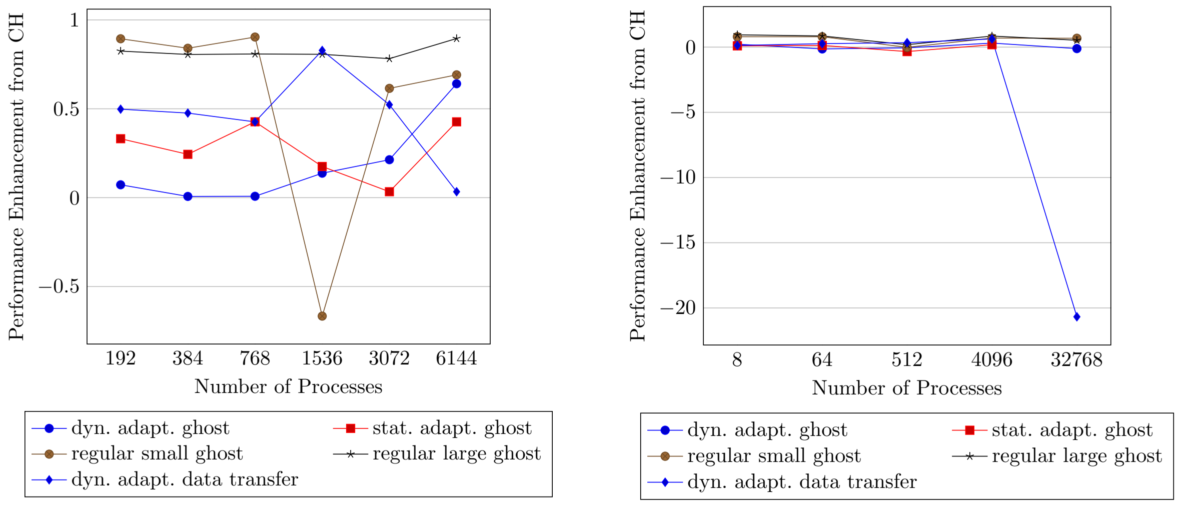


FIGURE 6.12 LBM scaling experiments: benefits of communication hiding (abbreviated as "CH") for strong scaling (left) and weak scaling (right) for operations involving communication. We compare maximum run times over all ranks and time steps for the respective operations.

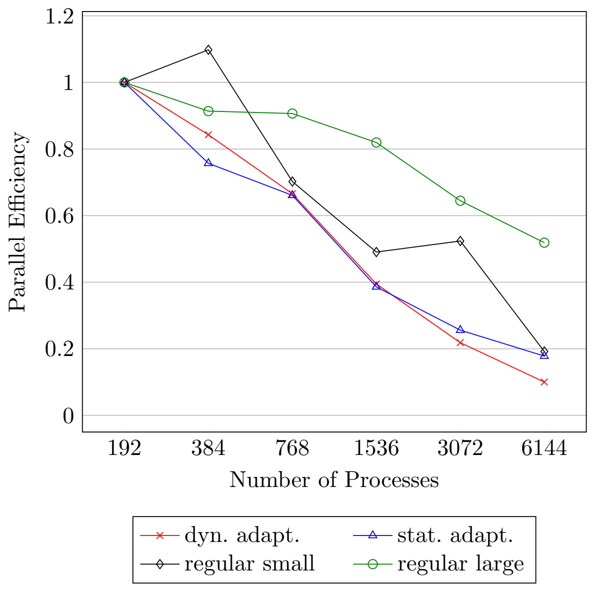


FIGURE 6.13 Overall parallel efficiency for strong scaling of the driven cavity scenario.

since choosing the optimal solution depends on several problem specific parameters that may evolve during run time.

Strong Scaling

In strong scaling experiments, we keep the total load constant and vary the number of available resources. We measure the parallel efficiency

$$\nu_{\text{strong}} = \frac{p_{\text{base}} \cdot t_{\text{base}}}{p \cdot t} \tag{6.5}$$

where p denotes the number of processes and t the simulation time. The optimal parallel efficiency is $\nu_{\text{strong}} = 1$, i.e., the increase in resources $p' = \frac{p}{p_{\text{base}}}$ equals the decrease in time to solution $t' = \frac{t}{t_{\text{base}}}$. Due to Amdahl's Law [214], the parallel efficiency is in general smaller than one.

We measure scaling from eight to 256 fully occupied nodes, i.e., 192 to 6144 processes, allocating one MPI rank per process. We use regular grids of level seven and eight as well as adaptive grids from level six to nine and compare their overall parallel efficiency in Fig. 6.13. The regular grids have approximately 2 and 16 million cells, the adaptive grids 8 million. In each case, approximately half the cells are boundary cells.

We plot the performance of our implementation, that is fluid lattice updates per second per core (FLUPSC), in Fig. 6.14. To this end, we consider the time steps on the maximum refinement level. We measure the time of the LBM solver and divide it by the number of fluid cells active during the respective time step. One step of the LBM solver

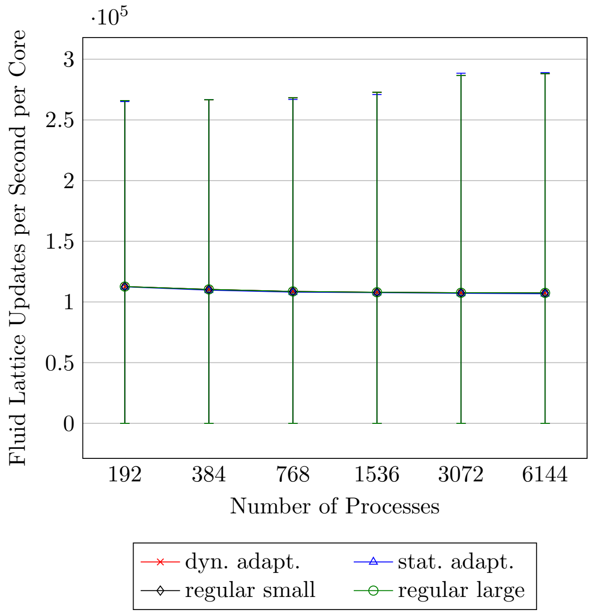


FIGURE 6.14 Fluid Lattice Updates per Second per core for strong scaling of the driven cavity scenario. The average value is the curve and the respective minimum and maximum values are shown as error bars. Empty processors, i.e., processors containing no active cells or only boundary cells, perform no fluid updates, thus they store zero.

contains collision and populating virtual cells, updating from virtual cells, streaming, swapping of the double-buffer, and exchanging messages with neighboring processes. Operations with virtual cells are only performed if they are necessary. The number of fluid cells per finest level time step changes due to multivariate time stepping and dynamic grid adaptivity including repartitioning. Empty processors, i.e., processors that only contain boundary cells, perform no fluid updates, thus, they store zero. We show the average value as the curve and the respective minimum and maximum values as error bars. We see almost no differences for the different discretizations and no drop in performance if the load per process decreases.

We further analyze the scaling behavior of our implementation by investigating the individual sub-steps of the algorithm. To this end, we measure the execution time of each main algorithmic step on each rank and take the maximum value over all ranks. To allow for potential grid adaptation, we measure 10×16 steps and average over the ten maxima arithmetically.

As the regular grid neither contains grid change nor LBM operations on different levels, it is the simplest case. We visualize the scaling of each operation in Fig. 6.15. Streaming is the dominant operation and both, collision and streaming run into a saturation effect where the load per process is too small for an efficient scaling. This happens between 768 and 1536 processes for collision and between 3072 and 6144 processes for

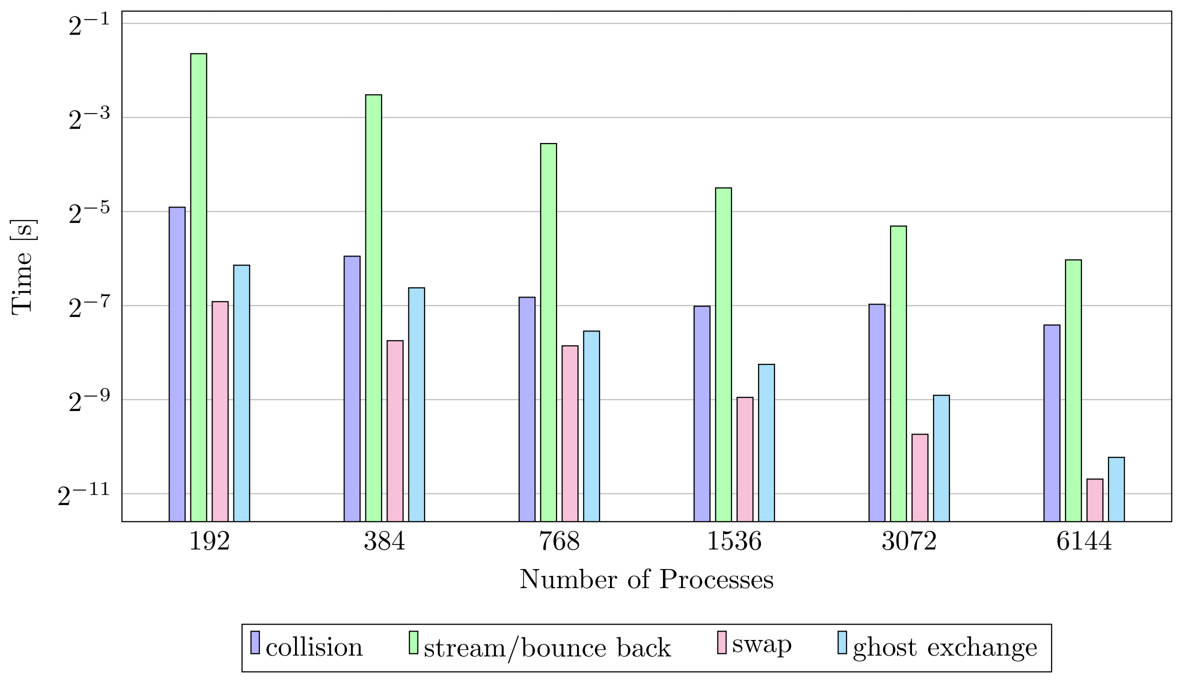


FIGURE 6.15 LBM driven cavity scalability test case: Strong scaling of LBM sub-steps for a regular grid of level eight. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation.

streaming.

We show the same plot for static adaptivity in Fig. 6.16. Compared to the regular case, we now have LBM operations on multiple refinement levels with multivariate time stepping. We add up the cost for each level, i.e., we do not weigh operations on higher levels according to multivariate time stepping. Instead of showing a full coarse time step, $\sum_{i=\ell_{\min}}^{\ell_{\max}} 2^{i-\ell_{\min}} t(i)$,we create all columns in the figures as a worst-case estimation of a fine time step to better compare them with the regular grid, $\sum_{i=\ell_{\min}}^{\ell_{\max}} t(i)$. As we do not change our partitioning strategy, we introduce a load imbalance. The run time is generally longer than in the regular case. Besides streaming, which is more expensive due to the more complicated neighbor lookup including virtual cells, exchanging ghost information is another expensive operation. Similar to the regular case, we see saturation effects if the number of cells per process drops, for collision between 384 and 768 processes and for streaming between 768 and 1536 processes.

When considering dynamic grid adaptivity, not much is changing in the LBM step (Fig. 6.17). Total computational cost, however, now not only stems from the main loop but also from adapting the grid. Fig. 6.18 visualizes the cost of the adaptivity scheme

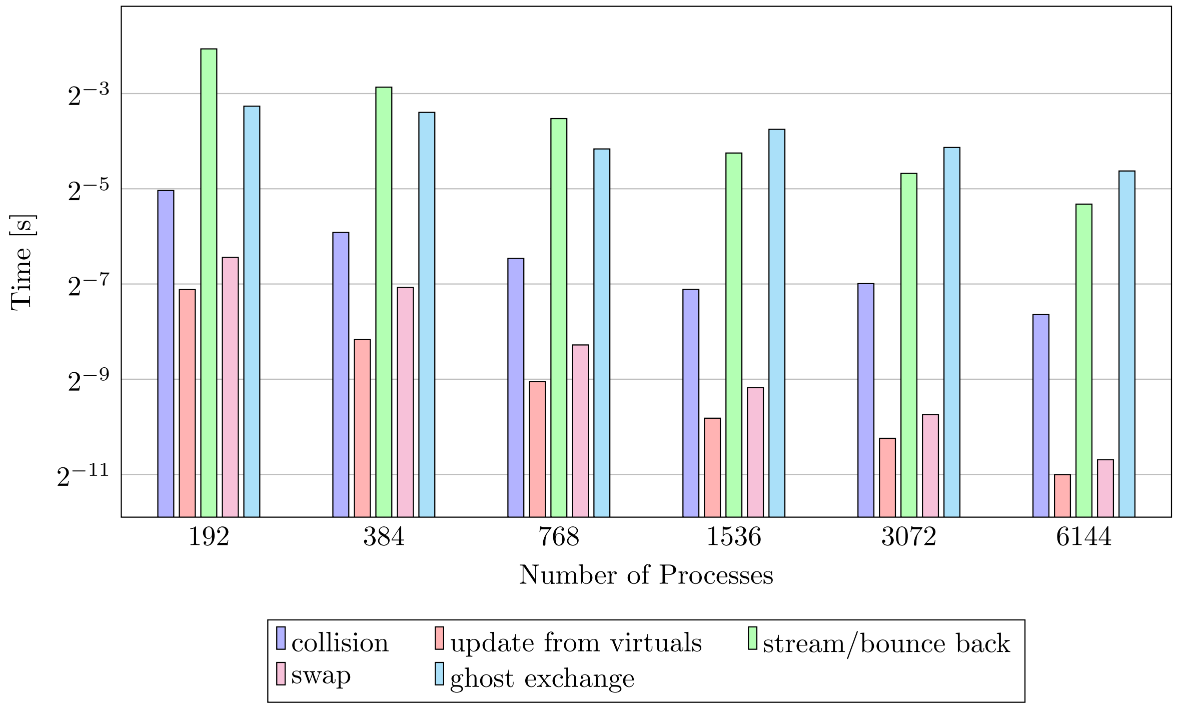


FIGURE 6.16 LBM driven cavity scalability test case: Strong scaling of LBM sub-steps for a statically adaptive grid of levels six to nine. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

explained in Sec. 5.2.3.

Here, the increase in building metadata is the most significant observation. When analyzing the individual operations in Fig. 6.19, it turns out that the issues stem from building the ghost layer, in particular from creating `p4est_ghost`.

Weak Scaling

In weak scaling experiments, we increase the total load with the number of available resources, thus keeping the load per resource constant. We measure parallel efficiency as

$$\nu_{\text{weak}} = \frac{t_{\text{base}}}{t} \tag{6.6}$$

where t_{base} denotes the time to solution of the initial run and t the time to solution of following scaling steps. Similarly to strong scaling, optimal scaling yields a parallel efficiency $\nu_{\text{weak}} = 1$. Thus, adding computational resources and increasing the problem-size such that the problem-size per resource is constant, should have no effect on the

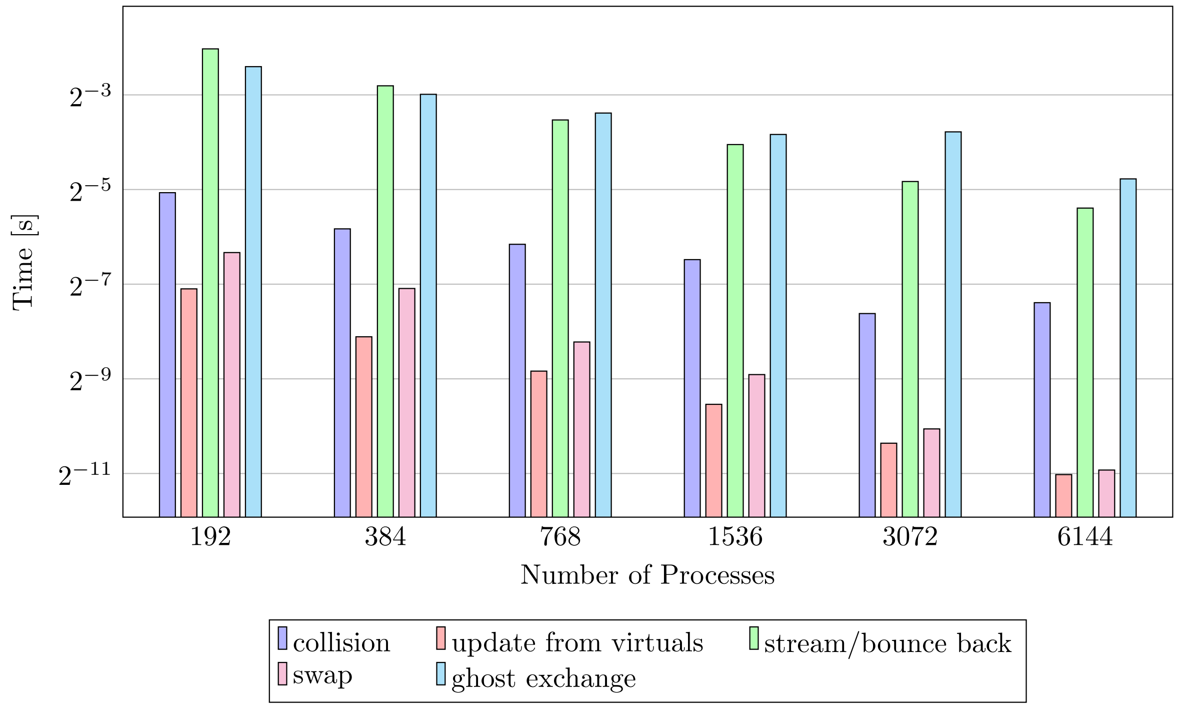


FIGURE 6.17 LBM driven cavity scalability test case: Strong scaling of LBM sub-steps for a dynamically adaptive grid of levels six to nine. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

overall performance. As Gustafson's Law [215] applies, it is, again, very hard to achieve optimal scalability.

We show upscaling results from one to 32,768 processes. We start at a given level ℓ and for each scaling step we increment the level by one and use eight times the computational resources. We use one MPI rank per processor and minimize the number of nodes by fully occupying $\lfloor \frac{p}{24} \rfloor$ processes. For the remaining $(p \mod 24)$ processes we add one node. Thus, we use 4096 nodes in the final scaling step.

We use regular grids with an initial refinement level of five and six as well as adaptive grids with initial levels from three to six. The regular grids have 32,768 and 262,144 cells per process.

In our setup, the load per process is not constant for adaptive grids, i.e., the number of cells does not continuously increase by a factor of eight, as shown in Fig. 6.20. In the adaptive scenarios, we initially have approximately 82,800 cells per process, 10,400 at 4096 processes, and 7750 at 32,768 processes. Thus, we initially have approximately twice the load per process compared to the first strong scaling step and end up with a load

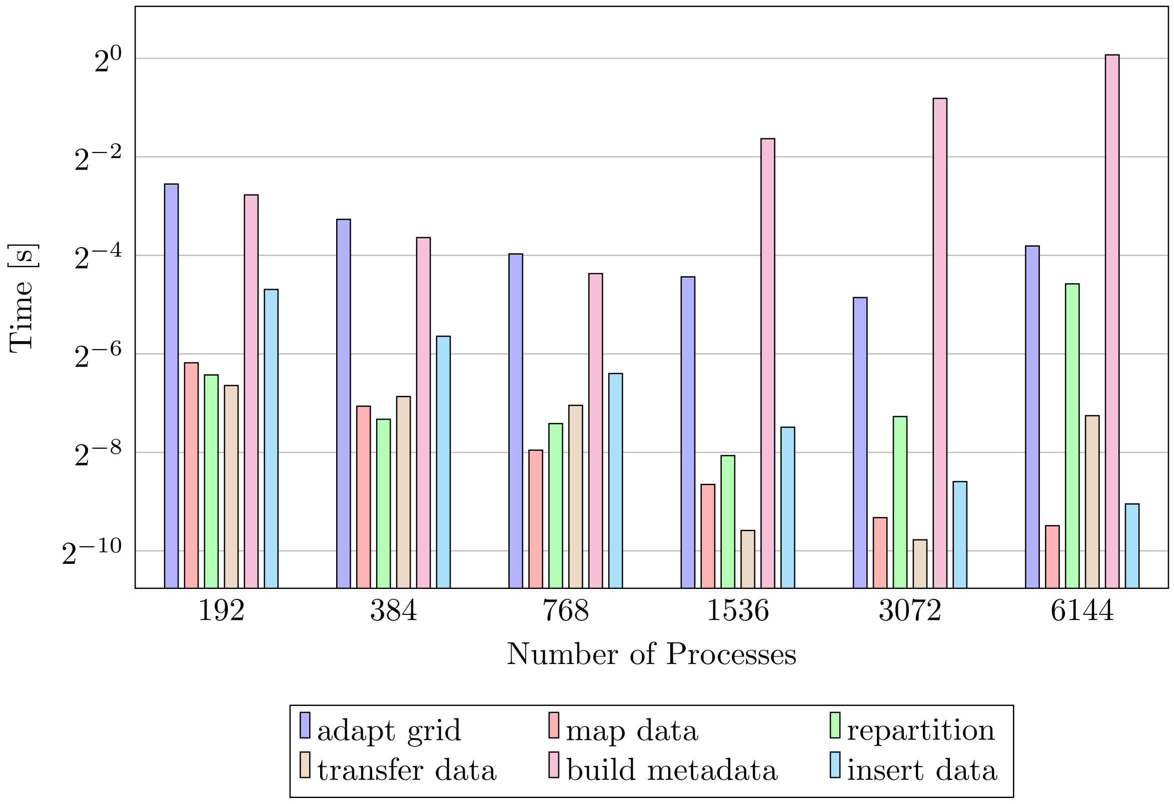


FIGURE 6.18 LBM driven cavity scalability test case: Strong scaling of sub-steps during grid change. For each operation, we show the arithmetic mean of the maxima over all processes.

per process that compares to strong scaling between 768 and 1536 processes. We mitigate this effect by normalizing the time by the number of relevant cells per process. Thus, for analyzing the LBM operations, we normalize by the average number of fluid cells, and for grid operations, we normalize by the average number of total cells. However, the effect of non-constant load cannot be fully compensated, because the load varies over a region that turned out to be the scaling limit in strong scaling. For the sake of completeness, we also show the absolute numbers in Appendix A.

The parallel efficiency for weak scaling of the driven cavity scenario using different discretizations is shown in Fig. 6.21.

We plot the performance of our implementation, that is fluid lattice updates per second per core (FLUPSC), in Fig. 6.22. Empty processors, i.e., processors that only contain boundary cells, perform no fluid updates, thus, they store zero. We show the average value as the curve and the respective minimum and maximum values as error bars. We see similar behavior for the regular grids and similar behavior for the adaptive grids.

As before, we split the 160 time steps into ten groups, take the maximum value from

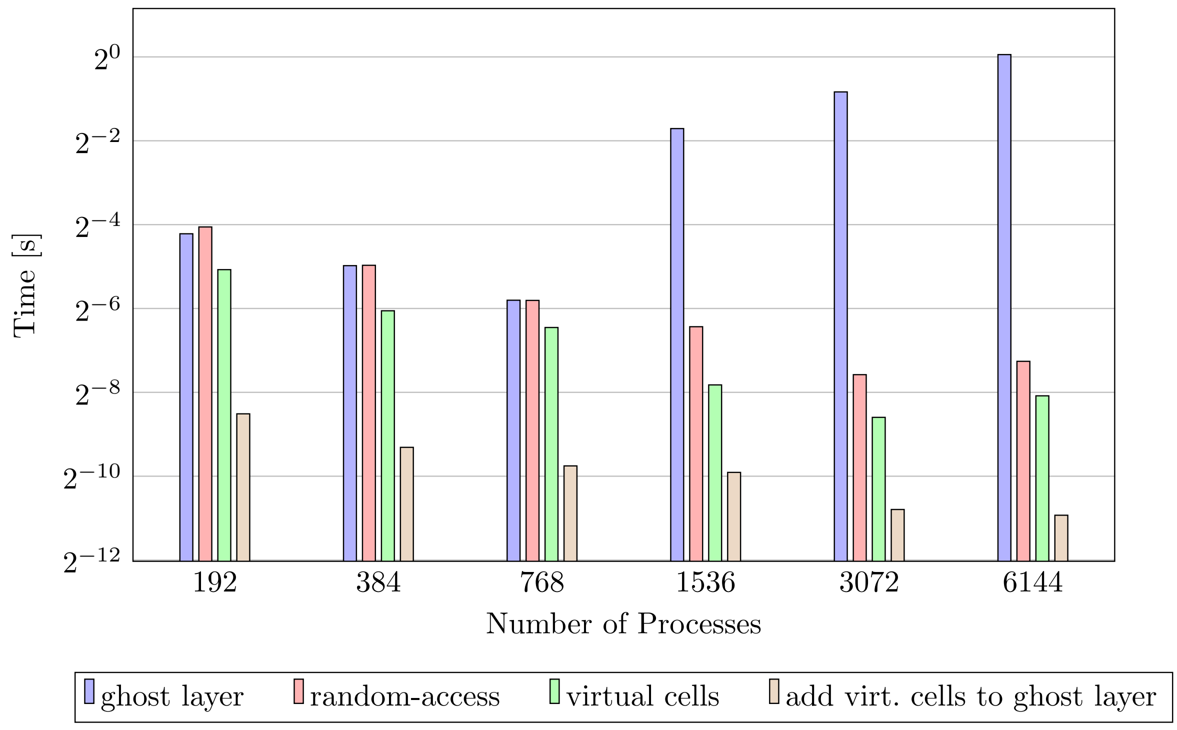


FIGURE 6.19 LBM driven cavity scalability test case: Strong scaling of sub-steps to build metadata during grid change. For each operation, we show the arithmetic mean of the maxima over all processes. We see that the significant increase in cost stems almost completely from building the ghost layer. The columns directly translate to creating the `p4est` components `p4est_ghost`, `p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`.

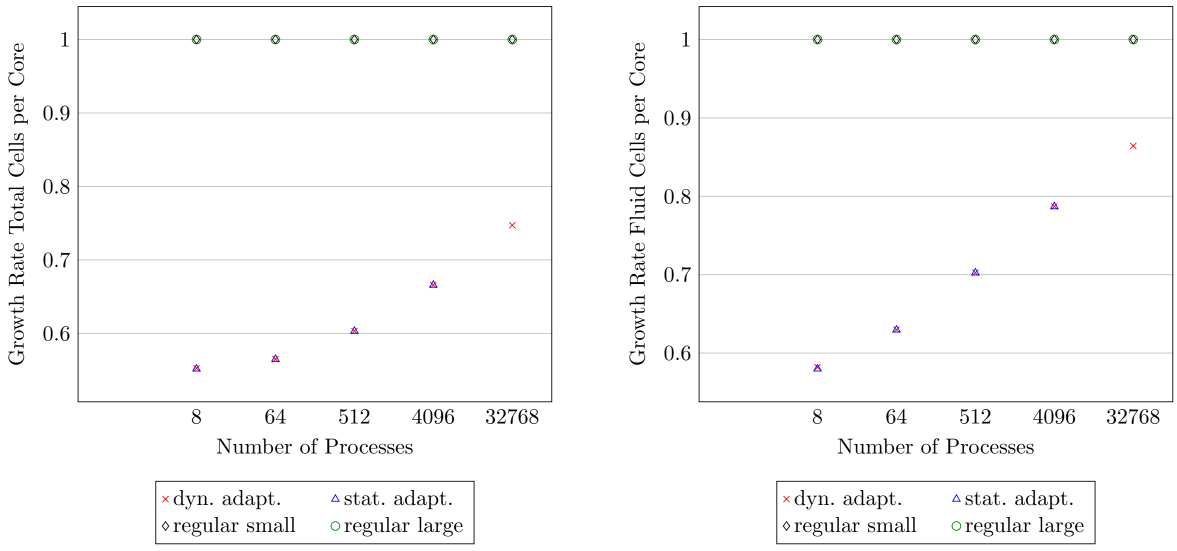


FIGURE 6.20 LBM driven cavity scalability test case: Load per process in weak scaling where each value compares to the previous scaling step in terms of total cells (right) and fluid cells (left). We see that for the adaptive grid load per process decreases, i.e., the number of cells does not increase by a factor of eight. We mitigate this effect normalizing the measured absolute times by the number of cells per process.

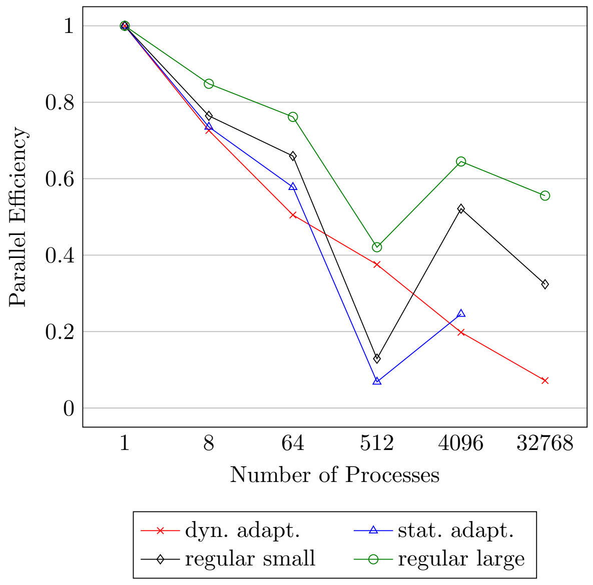


FIGURE 6.21 Overall parallel efficiency for weak scaling of the driven cavity scenario, weighted by the number of cells per process.

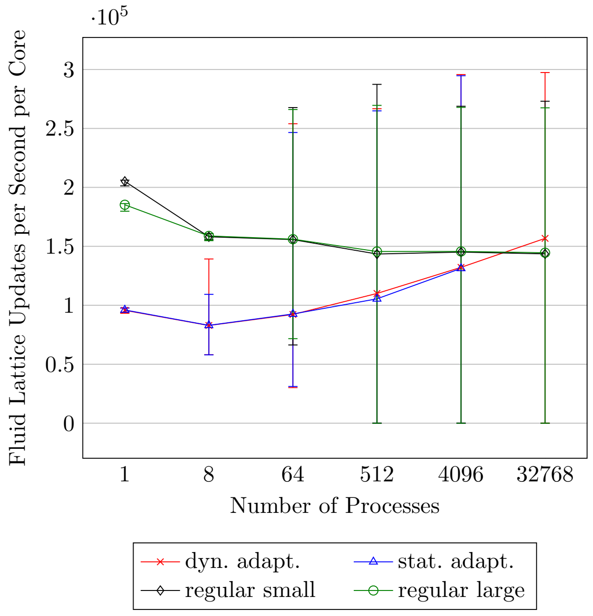


FIGURE 6.22 Fluid Lattice Updates per Second per core (FLUPSC) for weak scaling of the driven cavity scenario. The average value is the curve and the respective minimum and maximum values are shown as error bars. Empty processors, i.e., processors containing no active cells or only boundary cells, perform no fluid updates, thus they store zero.

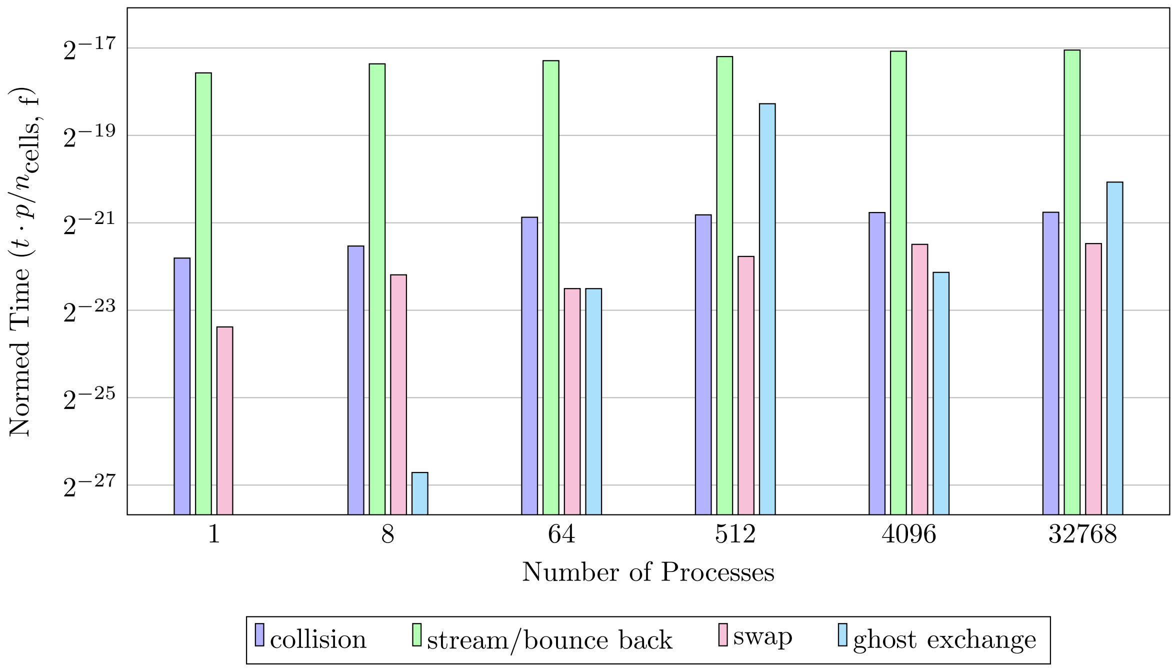


FIGURE 6.23 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a regular grid of level six. We split the overall 160 time steps into ten groups of sixteen steps. For each group and for each operation, we calculate the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation and normalize the time by the number of cells per process.

each group, and show their arithmetic mean. We proceed in the same way as we did for the strong scaling study: we begin with the main loop on a single level, then proceed to multiple levels before finally adding dynamic grid adaptivity. In Fig. 6.23, we show weak scaling of the LBM operations for a regular grid of level six. We observe that streaming is the dominant operation and find that the drop in performance at 512 processes stems from communication.

For the statically adaptive grid, we, again, find the streaming step to be the most costly operation. Again, we create the columns as $\sum_{i=\ell_{\min}}^{\ell_{\max}} t(i)$. Moreover, the drop in scalability at 512 processes, stems from exchanging data between processes[2]. As in the strong scaling study, computational cost in the adaptive case is slightly larger than in the regular case.

When adding dynamic grid adaptivity (Fig. 6.25), we see mostly the same effects. However, in the dynamically adaptive case, the communication issues do not occur for 512 cores.

For the grid adaptation step (Fig. 6.26) we, again, find a significant increase for building metadata. From Fig. 6.27, we see this is mostly generated from building `p4est_ghost`.

[2]The data for the run using 32,768 processes are corrupted and not shown here.

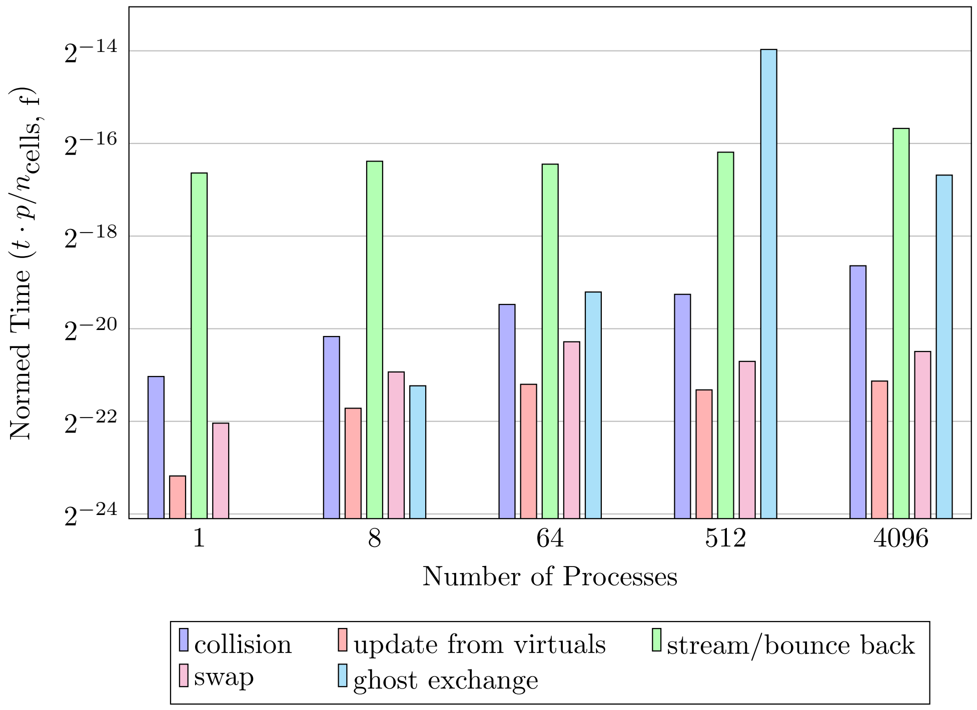


FIGURE 6.24 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a statically adaptive grid of levels three to six. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation and normalize the time by the number of cells per process. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

Additionally, transferring the payload to their new owners after repartitioning the grid shows a significant increase in the last scaling step. Both effects may stem from sending and receiving a rather large number of messages to and from many processes containing relatively little data.

Overall, we see good scalability in the `p4est` components touched in this project (`p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`). In the LBM implementation we suffer from our choice of using a collision optimized data layout, see Fig. 3.4. Here, we have greater flexibility for realizing dynamic spatial adaptivity (we could, e.g., change the grid and map data in the same loop) at the cost of more cache misses in the streaming step. Additionally, there were issues with building `p4est_ghost` which we could not fully resolve and that were surprising giving that `p4est_ghost` is known to scale well with low overall execution times [86]. In terms of performance, we have not expected to match the performance of specifically tuned implementations such as [54, 55] or [59, 60]. However, we propose testing an optimized algorithm with a streaming-optimized storage scheme on different hardware to resolve the issues with `p4est_ghost`. In the following,

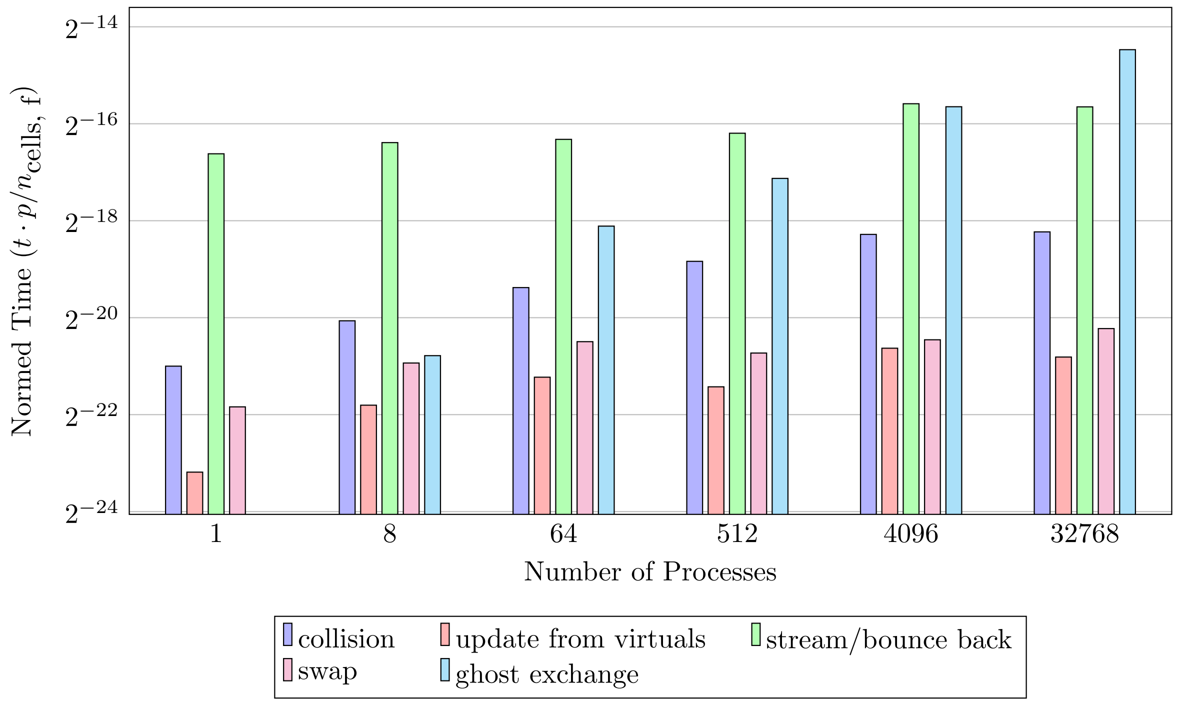


FIGURE 6.25 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a dynamically adaptive grid of levels three to six. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation and normalize the time by the number of cells per process. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

we analyze the scaling behavior of the Linked-Cell method and of coupled short-range molecular dynamics (MD) and LBM simulations.

6.2.2 Molecular Dynamics

In this section, we investigate the scaling behavior of the short-range MD implementation. Results from this section have previously been published in [4, 205].

We show the parallel efficiency of a static Lennard-Jones (LJ) weak scaling scenario. We arrange the particles in a grid such that they are equidistant and the system is perfectly balanced. Then, we calculate the forces between particles and reset it to zero before updating positions. With each scaling step we increase the number of particles and the volume of the simulation box such that the particle density is constant. We compare the previously existing implementation with Cartesian domain decomposition to our implementation using regular `p4est` grids in Fig. 6.28.

We see that we surpass the parallel efficiency of the original implementation at 96 processes. The absolute time-to-solution is faster for our implementation when using eight

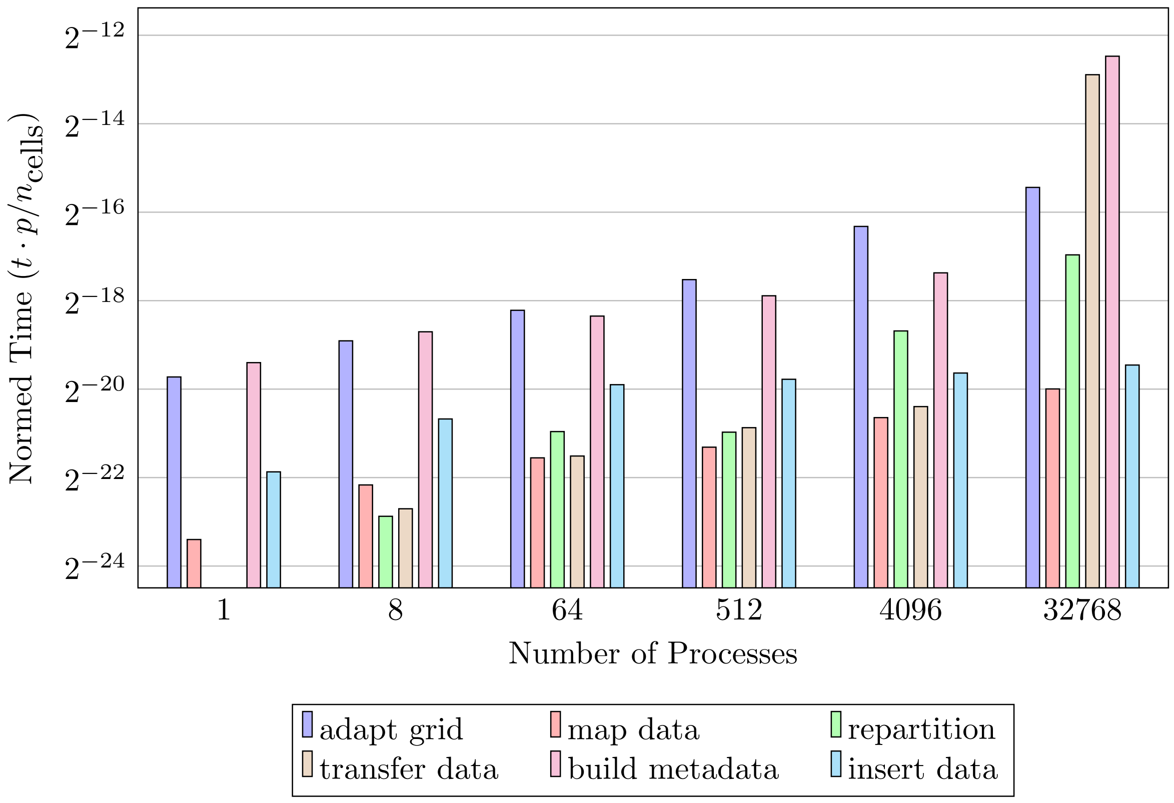


FIGURE 6.26 LBM driven cavity scalability test case: Weak scaling of sub-steps during grid change. For each operation, we show the arithmetic mean of the maxima over all processes. We normalize the time by the number of cells per process.

nodes (192 processes) or more. The main reason is the asynchronous communication that we have implemented for the SFC-based domain decomposition.

We show the benefits of dynamic repartitioning in Fig. 6.29. Here, we perform a simulation containing 1700 particles per process on 100 fully occupied nodes, i.e., 2400 processes. In total, we simulate 4,080,000 particles which in this scenario agglomerate to small droplets. We simulate 1,000,000 time steps and compare the cost of 1000 consecutive force calculations in the Linked-Cell algorithm as well as the imbalance between the processes for the unbalanced, default version of ESPResSo with our the dynamically repartitioned version based on `p4est`. The imbalance I is defined as

$$I = \frac{\max t}{\text{avg } t}, \tag{6.7}$$

where $\max t$ is the maximum run time for a specific operation and avg t the average. We repartition the simulation whenever the imbalance exceeds 1.1. Dynamic repartitioning saves computation time even in this simple scenario and effectively keeps the imbalance at a level close to one.

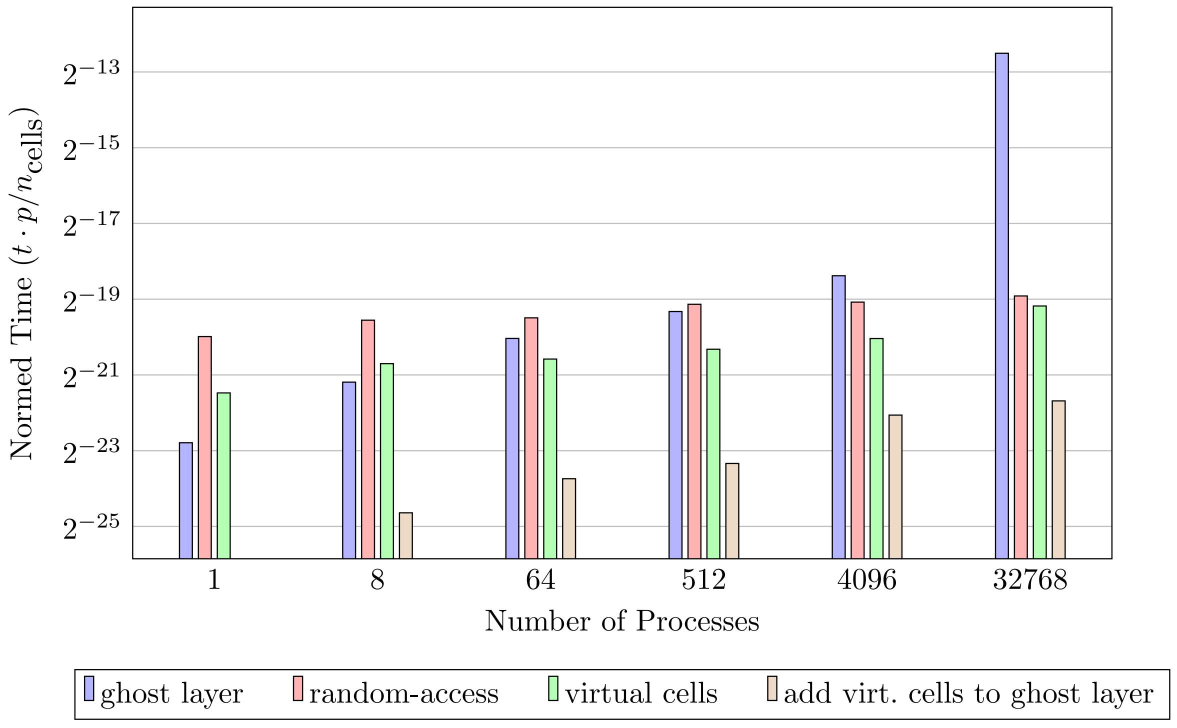


FIGURE 6.27 LBM driven cavity scalability test case: Weak scaling of sub-steps to build metadata during grid change. For each operation, we show the arithmetic mean of the maxima over all processes. We normalize the time by the number of cells per process. We see that the significant increase in cost stems almost completely from building the ghost layer. The columns directly translate to creating the `p4est` components `p4est_ghost`, `p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`.

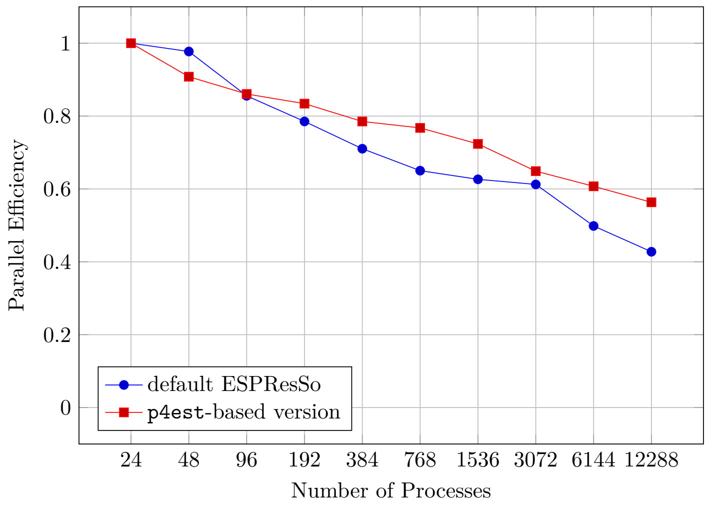


FIGURE 6.28 Weak scaling of short-range MD in a static LJ scenario. We compare the existing static domain decomposition with our SFC-based implementation using regular `p4est` grids.

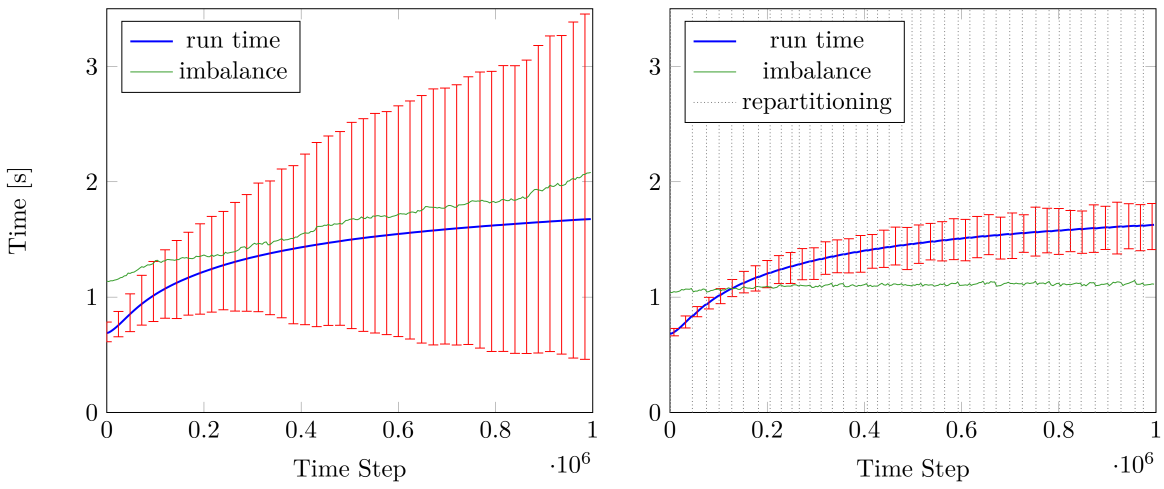


FIGURE 6.29 Short-range MD test case including dynamic repartitioning: Run time in seconds for 1000 consecutive time steps each and imbalance for the force calculation of 4,080,000 particles on 2400 processes. Left: default ESPResSo, right: SFC-based version using regular `p4est` grids and dynamic repartitioning. The error bars indicate the maximum and minimum run time, respectively, the blue line indicates the average. Note that the maximum run time determines the overall performance. The imbalance is shown in green. The vertical dashed lines in the right picture indicate time steps at which re-balancing is performed.

Hirschmann et al. compared this implementation using SFC-based partitioning to a graph-based partitioning scheme based on ParMETIS [217, 218] in [216]. They found SFC-based partitioning to be slightly slower when comparing different imhomogeneous MD scenarios and dynamic repartitioning. However, it has to be noted that fully coupled simulations using graph-based partitioning are infeasible with the current version of ESPResSo, because none of the discretizations of the electrokinetic equations is ported to a graph-based domain-decomposition. For regular grids, there are approaches based on padding [219], for adaptive grids it is questionable if this approach is feasible.

In the following, we analyze the scalability of our coupled implementation and show results towards the fully coupled electrokinetic system.

6.2.3 Coupled Simulations

In this section, we investigate the performance and scaling behavior of a coupled particle-fluid application using short-range MD and the LBM. Results presented in this section have previously been published in [206] and [207].

We demonstrate our coupling scheme and our algorithms for building the finest common tree (FCT) work by a weak scaling experiment using a rectangular domain with periodic boundaries in all directions with a side length of $2a \times a \times a$. Here, a denotes some length that we increase with each scaling step. We discretize the domain using two

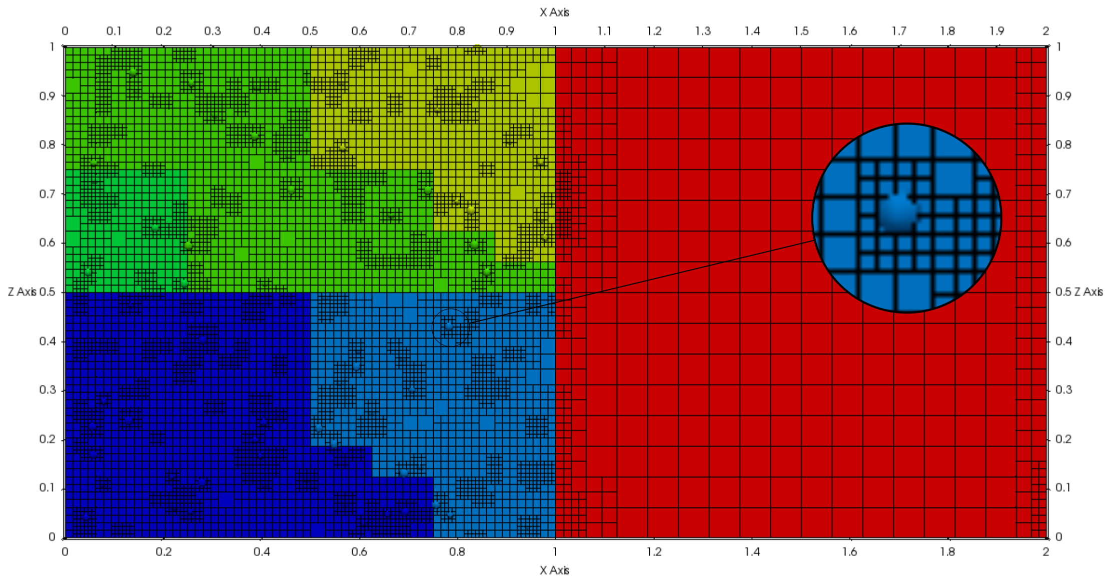


FIGURE 6.30 Setup and refinement pattern of the coupled short-range MD-LBM simulation without obstacle. We show a cut through the y-plane at $y = 0$. The system has periodic boundaries in all directions. We use a dynamically adaptive grid where we refine cells in the direct vicinity of particles. All other cells are implicitly coarsened. Different colors indicate different processes. For visualization purposes, we show fewer particles and a smaller number of ranks than in our scaling runs.

octrees. Within one half, we insert 1000 particles per process randomly with uniform distribution. Each particle has no velocity and is placed into a fluid-at-rest. We use a purely repelling LJ pair potential, i.e., we set the cut-off radius $r_c = 1$ and the parameters $\sigma = 1$ and $\varepsilon = 1$. We apply a constant force in x-direction. The fluid consists of four different levels, we refine cells around particles such that the coupling takes place at the maximum refinement level. This setup leads to particles moving in x-direction. To keep coupling on the finest grid level, we have to adapt the grid and repartition both particles and fluid[3]. To this end, we have to repeatedly build the FCT. We show the setup in Fig. 6.30.

We have performed two weak scaling runs, using the same scaling strategy as in Sec. 6.2.1: With each scaling step, we increment the minimum and maximum refinement level by one and use eight times the computational resources. Additionally, we add eight times more particles and set $a' = 2a$, thus, also scaling up the volume by a factor of eight, to keep the density constant. We compare one set of runs, where we have an initial level of refinement ranging from three to six, *case 3-6*, to another run, where we have an initial level of refinement from four to seven, *case 4-7*. In both test cases, we use the same volume

[3]This way, we ensure fewer interpolation errors and more frequent interactions between both systems. Additionally, particles are often either part of the region-of-interest or in the near vicinity.

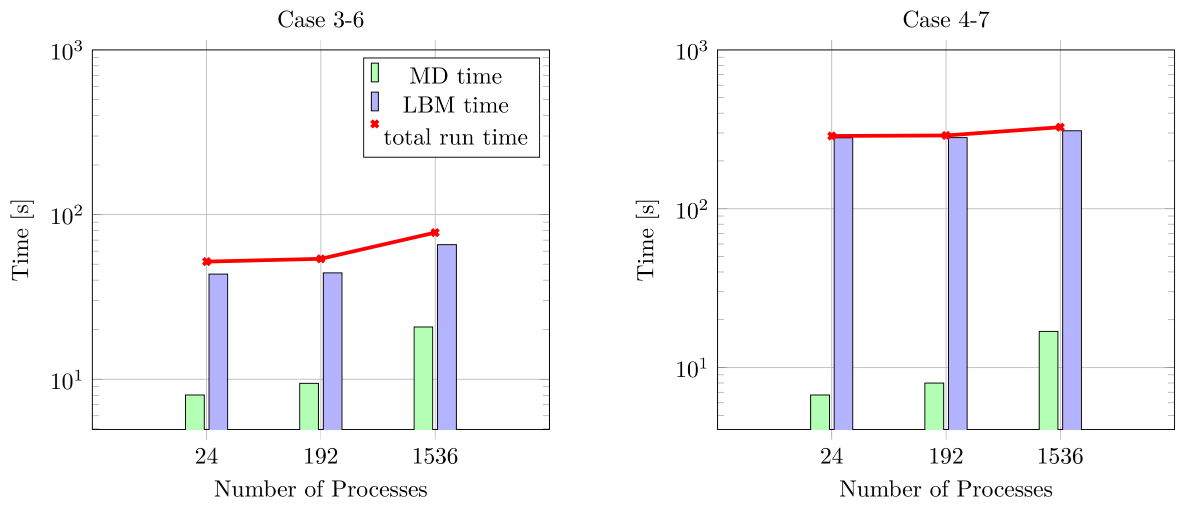


FIGURE 6.31 Weak scaling of the coupled short-range MD-LBM simulation without any obstacle. We compare the run time of short-range MD (green) and LBM (blue) to the total run time for our test cases.

and the same number of particles. We begin scaling for 24 processes and scale up twice to 1536 processes. We adapt the grid after sixteen time steps and run the simulation for a total of 160 time steps. For joint repartitioning using the FCT, we choose uniform weights for the LBM and weigh the cells of the Linked-Cell algorithm with the number of particles per cell. The computational times for LBM and short-range MD as well as the total time are shown in Fig. 6.31. We see a drop in overall performance of approximately 0.5 for *case 3-6* and of approximately 0.135 for *case 4-7*. Both losses stem from the respective sub-components. We obtain an imbalance I, as defined in Eq. (6.7), that is bounded by 1.16. The cost of building the FCT is only a small fraction of adapting the grid. For the largest test case (*case 4-7* on 1536 processes with approximately 120 million cells), the summarized run time of creating the FCT, Alg. 5.2, and jointly repartitioning the grids of the Linked-Cell algorithm and the LBM, Alg. 5.3, is approximately 0.053 s which is less than 2% of the total run time for adapting the grid.

We see good overall scalability. Particularly for the larger case, *case 4-7*, LBM dominates the run time. In order to move towards the target application, we added a simple pore model to the domain. This pore consists of two planes with a cylindrical hole and a rounded edge. We choose the same setup as before and add 1000 particles per process uniformly at random into one half of the domain. We illustrate the setup in Fig. 6.32.

We compare the run time of six different discretizations for the pore scenario: Default ESPResSo using a discretization resembling refinement levels six and seven, corresponding regular `p4est` grids of levels six and seven, and dynamically adaptive grids from initially level three to six and four to seven. When scaling up according to the same principles as

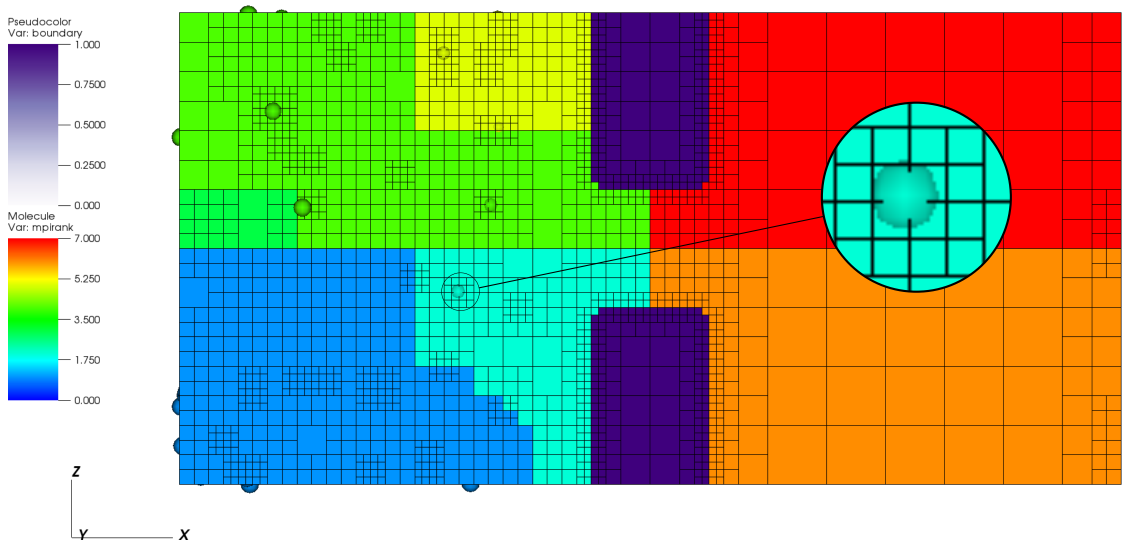


FIGURE 6.32 Setup and refinement pattern of the coupled short-range MD-LBM simulation with a simple pore obstacle. We show a cut through the y-plane at $y = 0.5a$. The system has periodic boundaries in all directions. We use a dynamically adaptive grid where we refine cells in the direct vicinity of particles and around the surface of the pore. All other cells are implicitly coarsened. Different colors indicate different processes. For visualization purposes, we show fewer particles and a smaller number of ranks than in our scaling runs.

stated above, we obtain results shown in Fig. 6.33. As the default version of ESPResSo only accepts rectangular sub-domains of exactly the same size, we scaled the default version from 16 to 1024 processes.

When comparing the run time of the regular `p4est` grid with the adaptive grid, we see an actual benefit of spatial adaptivity, because the adaptive implementation is faster than the regular grid. With the same resolution in the area of interest, i.e., around the particles, the total time to solution of the adaptive grid is approximately two-thirds of the regular case in the smaller scenario where we scale from level six. In the larger case, we can save approximately half of the total run time. Moreover, the imbalance induced into the run time dominating LBM algorithm due to the particles is generally bounded by 1.5 in both dynamically adaptive scenarios, except from the first MD step. We show the imbalances of the largest scaling step in Tab. 6.2 for the dynamically-adaptive discretization from level five to eight, based on the smaller adaptive case, and the dynamically-adaptive from level six to nine, based on the larger adaptive case, in Tab. 6.3.

Additionally, even in the largest scenario, a full cycle of adapting the grid and jointly repartitioning two `p4est` instances takes on average less than a second. Thus, even if in this scenario both `p4est`-based algorithms were slower than their counterparts in the default implementation, we can build scenarios, where spatial adaptivity pays off. In the

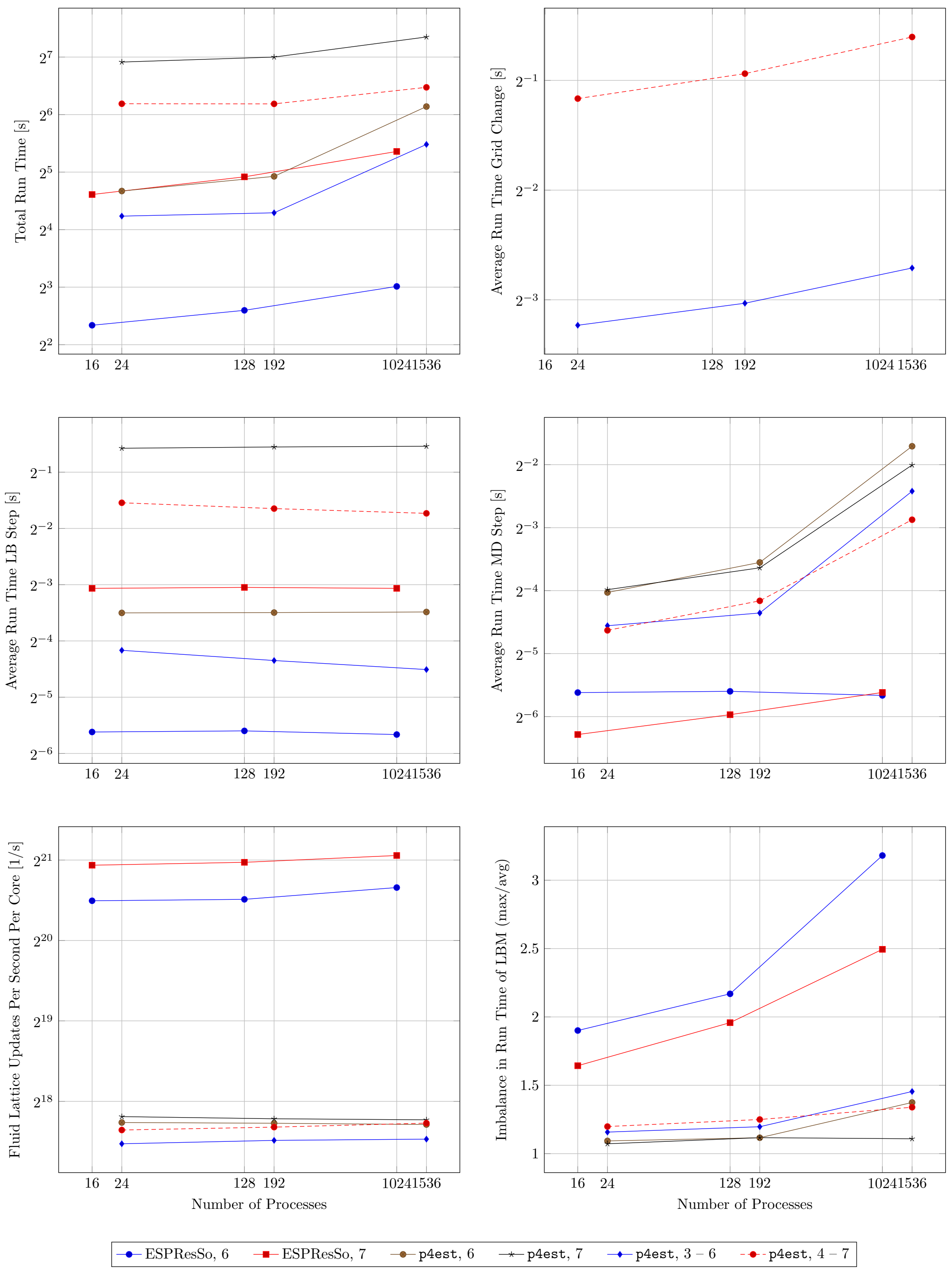


FIGURE 6.33 Weak scaling of the coupled short-range MD-LBM simulation with a simple pore obstacle. We compare the run time of six different discretizations for the pore scenario. We show the total run time (top left), the average run times for adapting the grid (top right), LBM algorithm (center left), and short-range MD (center right), FLUPSC (bottom left), and the imbalance for the LBM (bottom right).

TABLE 6.2 Weak scaling test case for the coupled particle-fluid simulation with a simple pore obstacle: We show the imbalances for every two coarse time steps, i.e., for each grid configuration, for a run on 1536 processes. We use a dynamically adaptive grid with refinement levels between five and eight. This is the last scaling step for the smaller test case. We measure imbalance as defined in Eq. (6.7).

grid version	n_{cells}	imb. LBM	imb. MD	imb. grid change
config. 0	$1.371 \cdot 10^7$	1.289	6.869	1.008
config. 1	$1.37 \cdot 10^7$	1.256	1.006	1.008
config. 2	$1.371 \cdot 10^7$	1.256	1.013	1.009
config. 3	$1.37 \cdot 10^7$	1.256	1.012	1.009
config. 4	$1.371 \cdot 10^7$	1.253	1.015	1.009
config. 5	$1.37 \cdot 10^7$	1.257	1.006	1.008
config. 6	$1.371 \cdot 10^7$	1.268	1.018	1.009
config. 7	$1.37 \cdot 10^7$	1.264	1.01	1.022
config. 8	$1.371 \cdot 10^7$	1.472	1.006	1.009
config. 9	$1.37 \cdot 10^7$	1.272	1.015	1.008

TABLE 6.3 Weak scaling test case for the coupled particle-fluid simulation with a simple pore obstacle: We show the imbalances for every two coarse time steps, i.e., for each grid configuration, for a run on 1536 processes. We use a dynamically adaptive grid with refinement levels between six and nine. This is the last scaling step for the larger test case. We measure imbalance as defined in Eq. (6.7).

grid version	n_{cells}	imb. LBM	imb. MD	imb. grid change
config. 0	$1.067 \cdot 10^8$	1.291	7.259	1.002
config. 1	$1.013 \cdot 10^8$	1.295	1.135	1.002
config. 2	$1.012 \cdot 10^8$	1.302	1.218	1.003
config. 3	$1.012 \cdot 10^8$	1.292	1.069	1.002
config. 4	$1.012 \cdot 10^8$	1.301	1.136	1.003
config. 5	$1.012 \cdot 10^8$	1.284	1.111	1.002
config. 6	$1.012 \cdot 10^8$	1.301	1.05	1.002
config. 7	$1.012 \cdot 10^8$	1.284	1.338	1.002
config. 8	$1.012 \cdot 10^8$	1.3	1.254	1.002
config. 9	$1.013 \cdot 10^8$	1.304	1.144	1.002

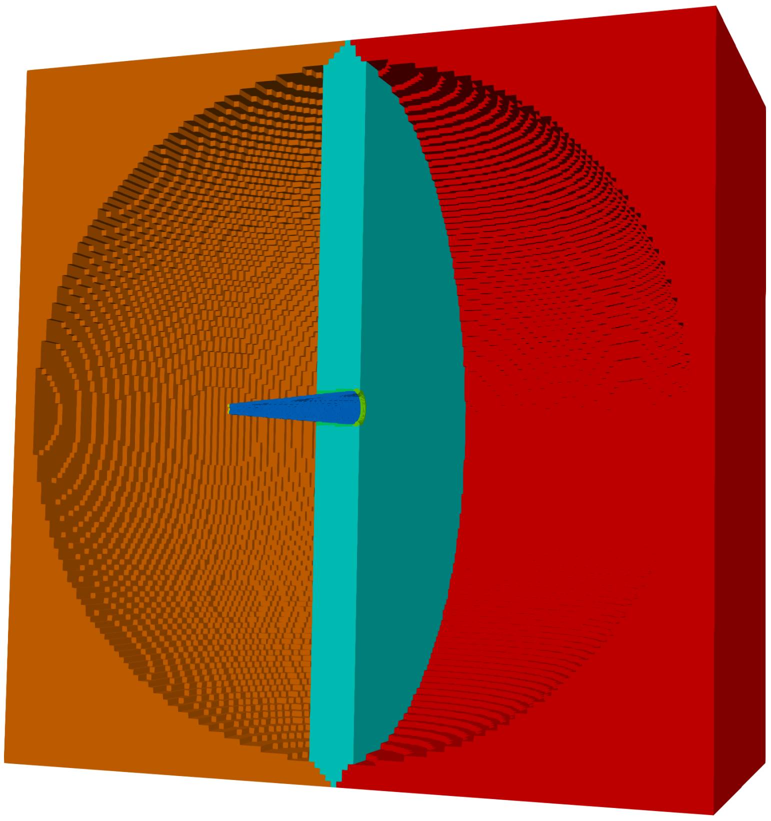


FIGURE 6.34 Simulation setup of a charged pore. We charge the body of the capillary (royal blue) and apply an electric field by setting a potential at the hemispheres (orange and red) to obtain an electro-osmotic flow around the pore.

following, we move closer to the target application by modeling a pore more precisely and adding wall charges as well as by simulating an ionic solution instead of an uncharged fluid.

6.3 Towards the Full Electrokinetic System

We have set up a three-dimensional geometry of the system described in [6, 26] and shown first simulation runs in [206]. A cut through the geometry is shown in Fig. 6.34.

The system's total size is $64\,\mu\text{m} \times 64\,\mu\text{m} \times 64\,\mu\text{m}$. We discretize the system by a single octree and use a discretization level from seven to ten. We statically refine the area around the pore, resulting in a grid consisting of 2,261,736 cells. We model the nanopore geometry as two hemispheres separated by a cylindrical plane with a hole. This hole is connected to a cone whose edges are rounded by a torus. The hemisphere's radius is set to $r = 30\,\mu\text{m}$, the total length of the pore is set to $L = 20\,\mu\text{m}$. The cone angle is set to $\alpha = 5°$, the thickness of the wall splitting both basins is $3\,\mu\text{m}$. In our first simulations, we used a larger pore diameter than in the real-world system, $r_{\text{tip}} = 250\,\text{nm}$. The real-world system has pore diameters r_{tip} ranging from 7.5 nm to 150 nm. To fully capture this, we need a finer grid: If we refine the grid up to level 16, we obtain a grid-spacing of approximately 1 nm. A regular octree grid of level 16 contains approximately $2.815 \cdot 10^{14}$ cells. We can give an upper bound for an adaptive grid with a level difference of four from our simulation up to level 10 as $2{,}261{,}736 \cdot 8^6 = 5.929 \cdot 10^{11}$. Compared to the regular grid with a memory footprint from $\mathcal{O}(1\,\text{PB})$, we end up in the regime of $\mathcal{O}(1\,\text{TB})$. If we can keep the minimum refinement level constant, we can reduce the memory footprint even more: we obtain $6.057 \cdot 10^8$ cells which reduces memory requirements to $\mathcal{O}(1\,\text{GB})$. However, it has to be noted, that the successive over-relaxation (SOR) solver takes very long to converge in these scenarios. An implementation of a parallel algebraic multi-grid method [220] can mitigate the problem, implementations based on `p4est` [221] have been scaled up to 122,880 processes [85].

We set the hemisphere potentials to $+1$ and -1. At the capilar body, we set a fixed charge density of $\varrho = -0.06$. The system contains two ionic species, ions and counterions which only differ in their valency (counterions: $z = -1$, ions: $z = 1$). The species' density is set to $c = 0.006$, and we choose the diffusion coefficient $D = 0.006$. We set the fluid density to $\rho = 26.106$, the kinematic viscosity to $\eta = 2.73$, the friction coefficient to $\Gamma = 4.3$, and the Bjerrum length to $\ell_B = 0.709$.

We show the resulting flow field after 8000 time steps with a fixed length of $\Delta t = \frac{1}{32}$ in Fig. 6.35.

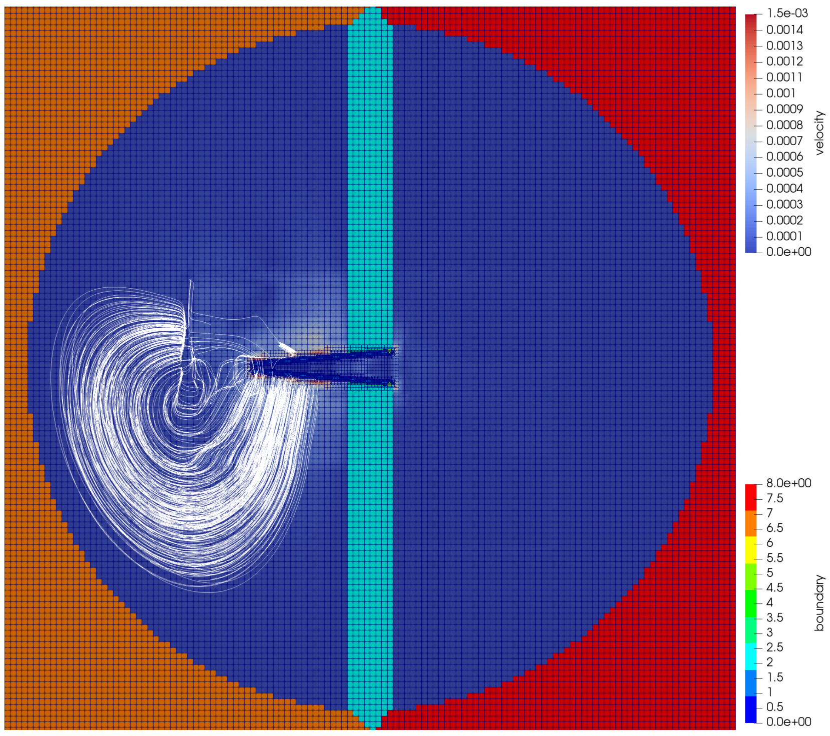


FIGURE 6.35 Velocity field of the simulated electro-osmotic flow in the charged pore system after 8000 time steps.

7 Conclusion

7.1 Contributions

In this work, we have presented our model for integrating dynamically-adaptive tree-structured grids into existing applications with minimal invasiveness. We have extended the `p4est` grid library such that it supports random-access from each grid cell to every neighboring cell. This allows reusing the numerical kernels in the application algorithms with the same data-access patterns but a different order of traversing the grid and a different domain-decomposition in parallel simulations.

To deal with refinement boundaries, we have integrated virtual cells. They allow considering a refinement boundary as an overlap region between two logically regular grids. This allows numerical kernels to exchange information with cells of the same size. To combine the information on both grids, we need a separate interpolation and restriction step.

This way, we have ported the discretizations of four different algorithmic components in ESPResSo, hydrodynamics, ionic flux, electrostatic, and molecular dynamics, one after another, to a `p4est`-based spatial discretization. To ensure process-local coupling during parallel execution, we have proposed a joint partitioning of multiple `p4est` instances with the same macrostructure, i.e., a matching set of octrees, and arbitrary microstructure, i.e., arbitrary refinement within each octree, using the finest common tree (FCT). We have shown at a small example using three processes and at two coupled simulations using 1536 processes that the introduced load imbalances are moderate.

We have tested our implementation in terms of physical correctness as well as in terms of performance and scalability. The simple test cases showed good agreement with the analytical solutions. For performance and scalability, we have shown the current state and proposed ways to optimize the implementation.

Additionally, we have taken important steps towards simulating the target application. We have extended the two-dimensional model of Rempfer et al. [6] to three dimensions and performed preliminary simulations using a statically adaptive grid and a larger pore diameter compared to the real-world application. Moreover, we have demonstrated that our setup reduces the memory footprint by six orders of magnitude compared to a corresponding regular grid. This makes the application accessible even on mid-range supercomputers.

7.2 Report on Minimal Invasiveness

At the end of this project, where we could simulate a preliminary version of the target system, we have changed the code in several places. According to cloc[1], we have added or changed around 20,000 lines of code compared to the last merge from the ESPResSo upstream branch. We present a detailed report, separated by programming language, in Tab. 7.1. Besides the algorithms already presented here, the added functionality includes (i) code to control `p4est` such as evaluating criteria for adapting the grid; (ii) managing the payload; i.e., allocation and deallocation or mapping data between grids for dynamic spatial adaptivity; (iii) utility functions regarding spatial coordinates in `p4est`-based grids, e.g., finding the respective cell given a position in $\mathbb{R}^3$; (iv) functions enabling dynamic load balancing of short-range molecular dynamics (MD) according to user-defined metrics [205, 207, 216], and (v) a graphics processing unit (GPU) implementation of the `p4est`-based lattice-Boltzmann method (LBM) using multiple GPUs, including VTK output [222].

Summarizing, we have preserved the project's core functionality and added new features. Some of these new features are independent of the grid implementation at hand, e.g., mappings between cells of different refinement levels during an adaptivity step or the logic for dealing with refinement boundaries. Other functionality had to be added, because the previous algorithm did not allow directly porting it to adaptive grids, e.g., replacing the discrete Fourier transform (DFT) solver by an successive over-relaxation (SOR) solver for solving Poisson's equation to calculate the elelectrostatic potential. However, we avoided implementing new numerical kernels for LBM, short-range MD, and the ionic flux: instead, we modified the data-access, integrated the functionality for dealing with refinement boundaries, and reused the existing logic.

[1]`https://github.com/AlDanial/cloc`

TABLE 7.1 Report on minimal-invasiveness: Code changes in ESPResSo in this project according to the cloc software.

Language	files	comment	code
C++			
same	0	5784	24029
modified	63	43	1059
added	16	1361	13434
removed	8	150	342
C/C++ Header			
same	0	4712	6524
modified	68	14	183
added	33	1621	3270
removed	8	120	194
CUDA			
same	0	1129	7803
modified	6	1	71
added	1	93	1048
removed	1	5	44
Cython			
same	0	1274	2405
modified	7	0	11
added	2	97	433
removed	0	1	25
Others			
same	0	2094	5774
modified	49	10	110
added	8	183	965
removed	2	78	291
SUM			
same	0	14993	46535
modified	193	68	1434
added	60	3355	19150
removed	19	354	896

7.3 Outlook

The next step following the work in this project should prevent that the implementation for the regular grid and the `p4est` grid may diverge in the future. We think, this is especially important in the time of decentralized distributed version control systems such as git. To further enhance efficiency, apart from the optimization we already mentioned, that is using a streaming-optimized data-layout in the lattice-Boltzmann method (LBM) (Fig. 3.4) and replacing the successive over-relaxation (SOR) solver of Poisson's equation for the electrostatic potential with an algebraic multigrid implementation, we propose expressing the numerical kernels independent of the underlying discretization. We can achieve this either by integrating a pre-defined framework such as dune-grid [223, 224] or by developing a domain-specific language, specifically tailored for the situation at hand.

Moreover, we can improve the LBM implementation numerically, by using a more robust collision kernel such as cumulant LBM [136, 137] and improve the accuracy at refinement boundaries by adding an interpolation step [59, 172, 173]. We can reduce the memory footprint by switching to a single-buffering scheme [139, 140].

To enhance overall performance, we propose following an ambitious vision: the complexity of the application at hand allows adding a hybrid parallelization approach to the MPI based distributed memory parallelization. Nodes of modern day supercomputers using graphics processing unit (GPU) as accelerator hardware typically combine between 20 and 60 central processing unit (CPU) cores with less than ten GPU. Examplary systems with this kind of node-architecture are TSUBAME 3.0 (28 cores, 4 GPU), Summit (44 cores, 6 GPU), Sierra (44 cores, 4 GPU), or Piz Daint (12 cores, 1 GPU). In our application, we can use as many MPI ranks as there are GPUs for the algorithms involved in solving the electrokinetic equations. This architecture uses patches, i.e., small regular grids, in `p4est` cells [192, 225]. A first version using regular grids was integrated in ESPResSo in [222]. We can use shared-memory parallelism on the remaining CPU cores using, e.g., OpenMP[2] or Intel TBB[3], to perform the molecular dynamics (MD) time step.

[2]https://www.openmp.org/
[3]http://threadingbuildingblocks.org/

A LBM Weak Scaling, Absolute Times

Absolute timing values for the lattice-Boltzmann method (LBM) driven cavity test case, weak scaling (Sec. 6.2.1) of the LBM sub-steps for a regular grid of level six (Fig. A.1), a statically adaptive grid (Fig. A.2) and a dynamically adaptive grid (Fig. A.3) with levels three to six. We show the absolute run times for grid change in Fig. A.4 and, specifically for constructing metadata, in Fig. A.5.

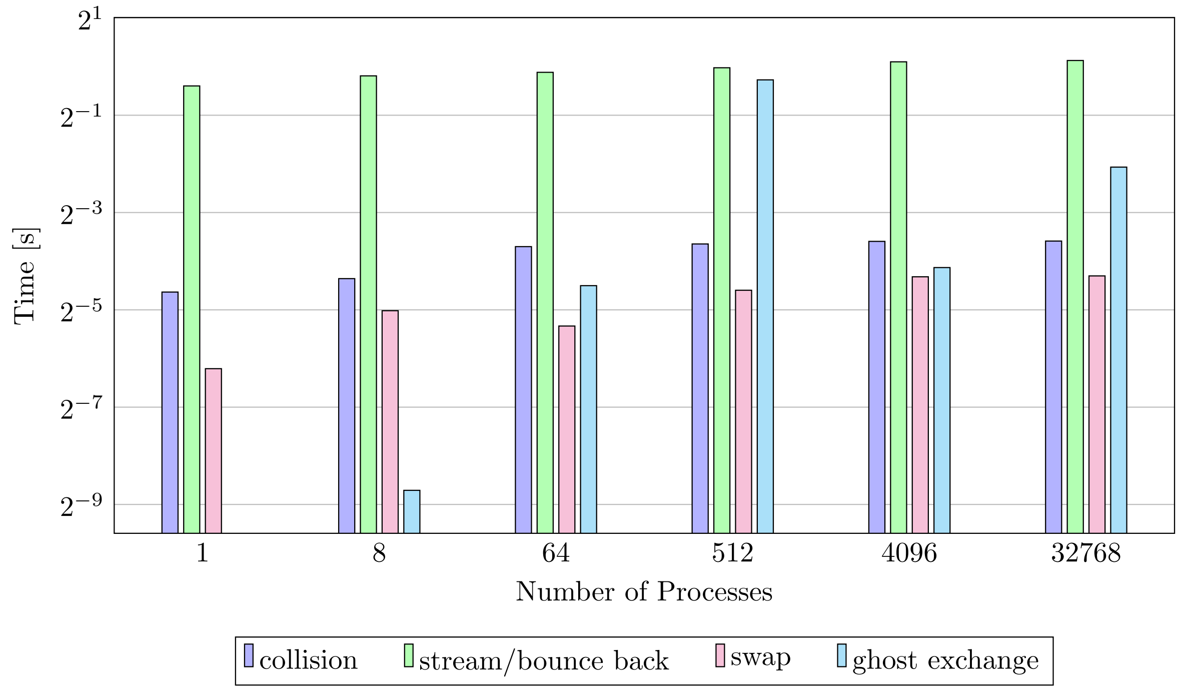


FIGURE A.1 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a regular grid of level six. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation.

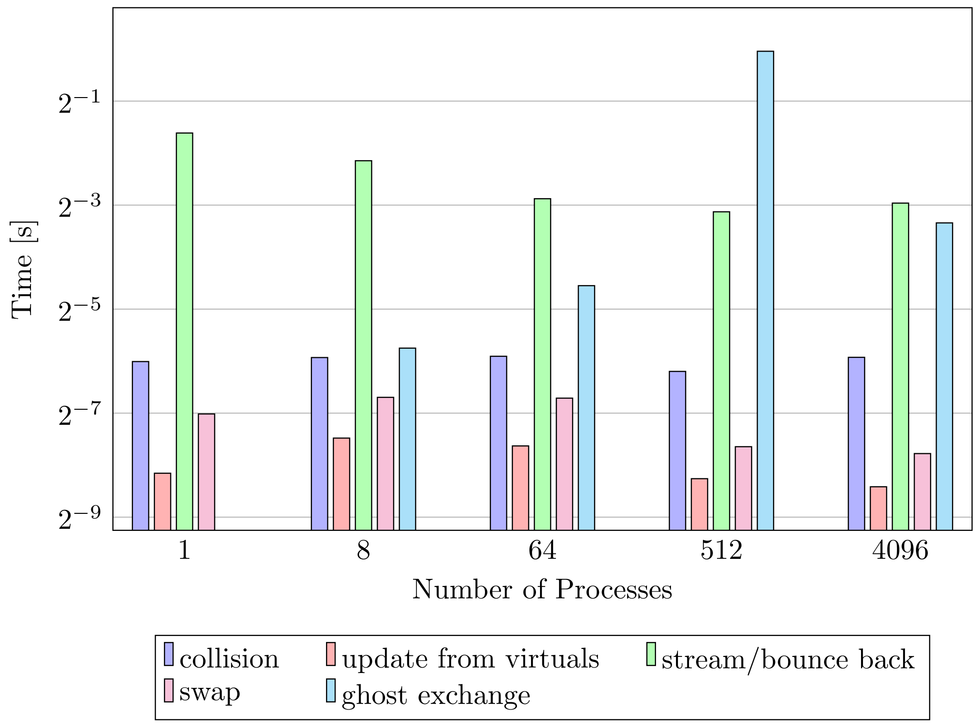


FIGURE A.2 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a statically adaptive grid of levels three to six. We split the overall 160 time steps into ten groups of sixteen steps. For each group we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

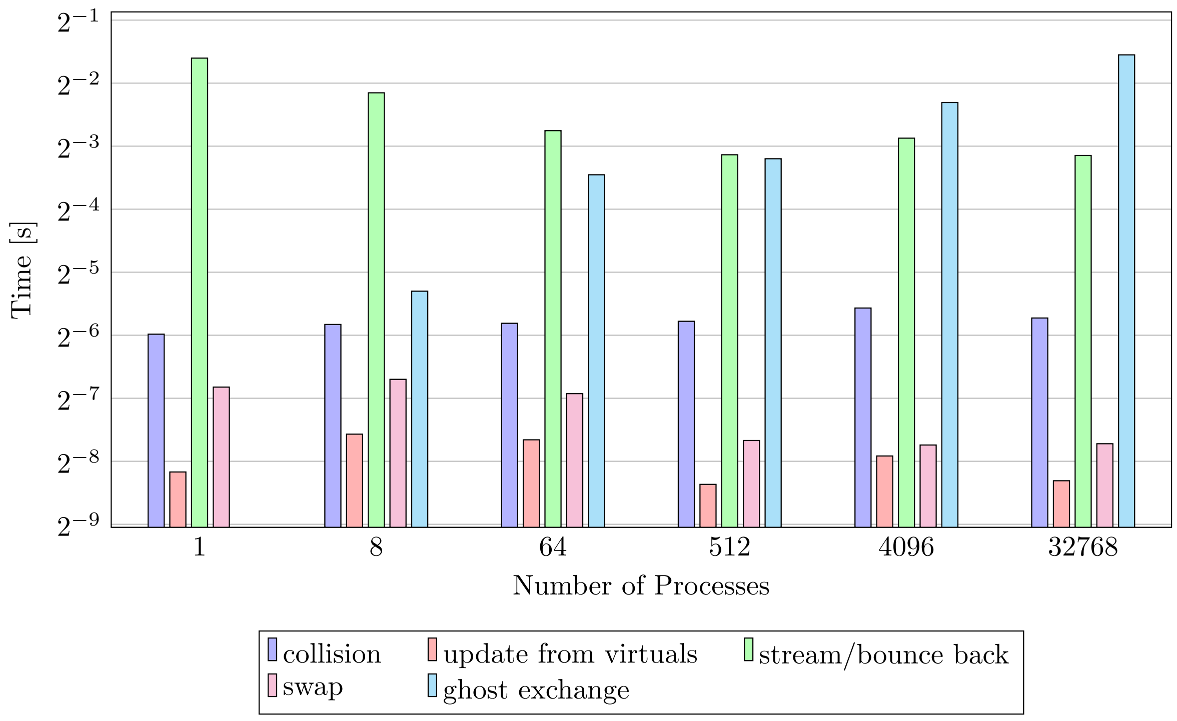


FIGURE A.3 LBM driven cavity scalability test case: Weak scaling of LBM sub-steps for a dynamically adaptive grid of levels three to six. We split the overall 160 time steps into ten groups of sixteen steps. For each group, we calculate for each operation the maximum value over all sixteen time steps and all processes. We show the arithmetic mean of these ten values for each operation. Values of multiple levels are summed using uniform weights, i.e., we do not include multivariate time stepping.

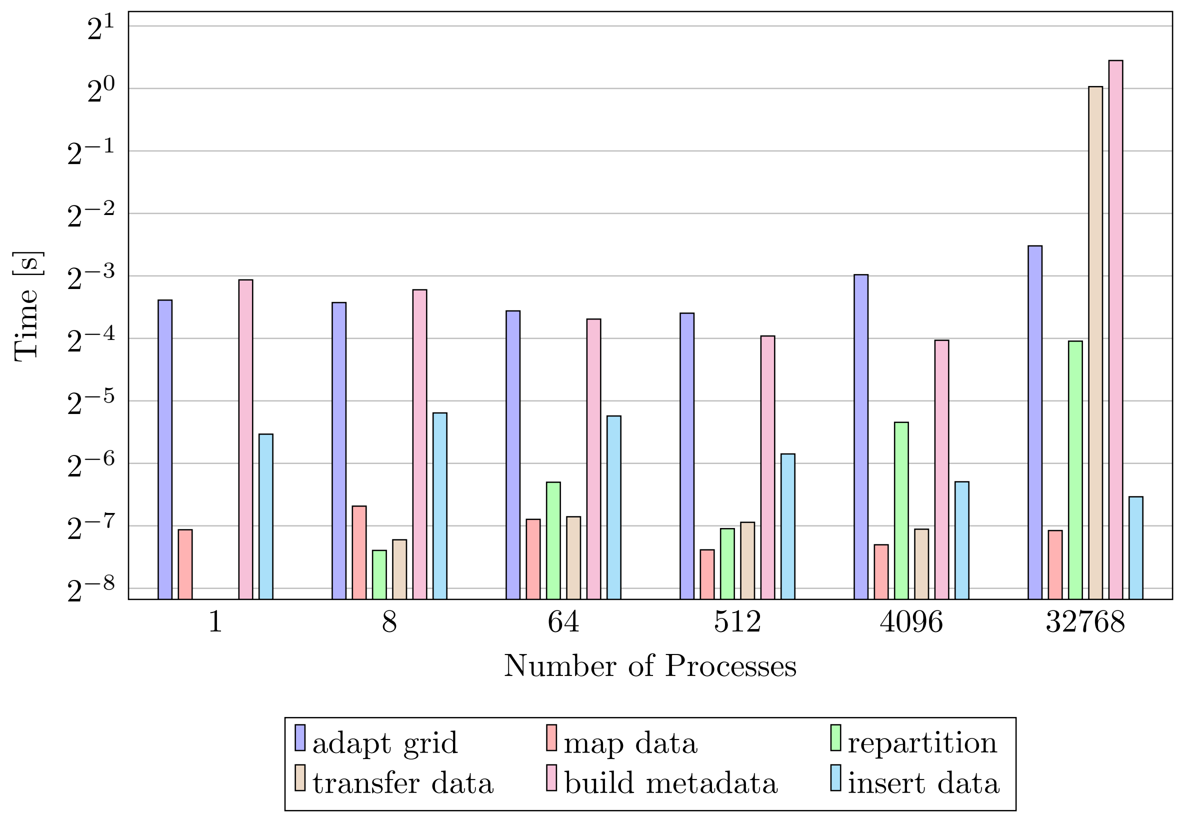


FIGURE A.4 LBM driven cavity scalability test case: Weak scaling of sub-steps during grid change. For each operation, we show the arithmetic mean of the maxima over all processes.

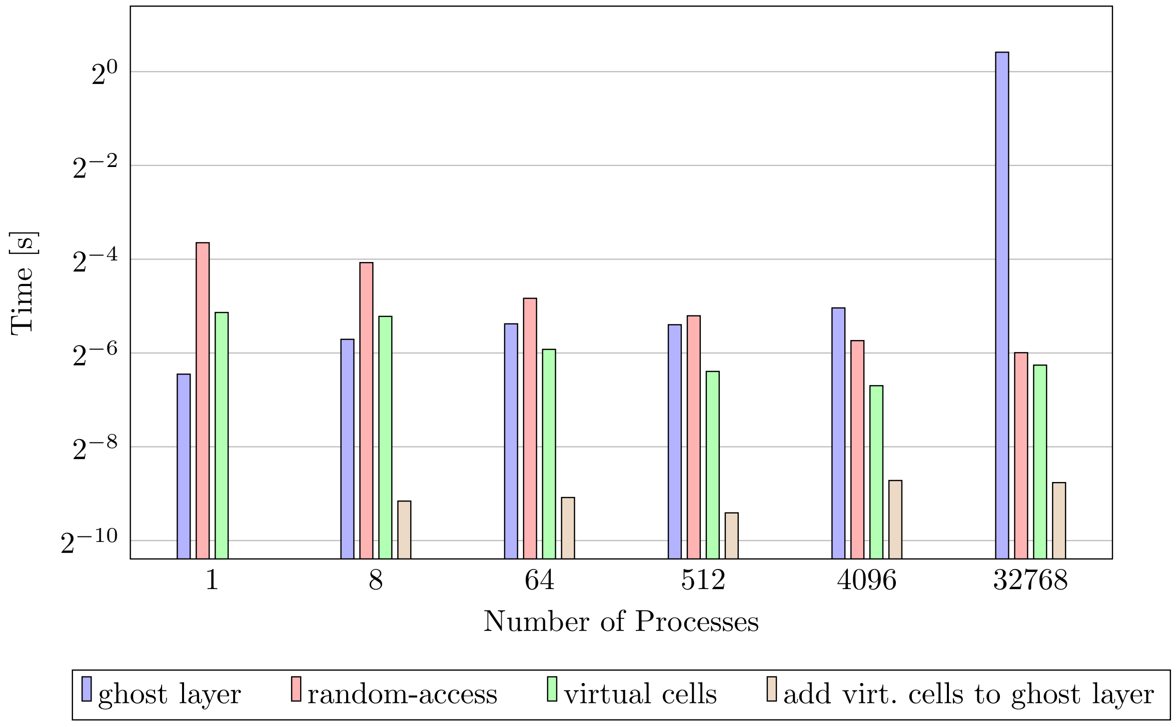


FIGURE A.5 LBM driven cavity scalability test case: Weak scaling of sub-steps to build metadata during grid change. For each operation, we show the arithmetic mean of the maxima over all processes. We see that the significant increase in cost stems almost completely from building the ghost layer. The columns directly translate to creating the `p4est` components `p4est_ghost`, `p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`.

B Bibliography

[1] M. Lahnert; C. Burstedde; C. Holm; M. Mehl; G. Rempfer; F. Weik. “Towards Lattice-Boltzmann on Dynamically Adaptive Grids – Minimally-Invasive Grid Exchange in ESPResSo.” *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*. Ed. by M. Papadrakakis; V. Papadopoulos; G. Stefanou; V. Plevris. ECCOMAS, 2016.

[2] M. Lahnert; T. Aoki; C. Burstedde; M. Mehl. “Minimally-Invasive Integration of p4est in ESPResSo for Adaptive Lattice-Boltzmann.” *The 30th Computational Fluid Dynamics Symposium*. Japan Society of Fluid Mechanics, 2016.

[3] M. Lahnert; C. Burstedde; M. Mehl. “Scalable Lattice-Boltzmann Simulation on Dynamically Adaptive Grids.” *to be submitted* (2019). Forthcoming.

[4] M. Brunn. “Coupling of Particle Simulation and Lattice Boltzmann Background Flow on Adaptive Grids.” Master’s thesis. Universität Stuttgart, 2017.

[5] I. Tischler. “Implementing adaptive Electrokinetics in ESPResSo.” Master’s thesis. University of Stuttgart, 2018.

[6] G. Rempfer; S. Ehrhardt; C. Holm; J. de Graaf. “Nanoparticle Translocation through Conical Nanopores: A Finite Element Study of Electrokinetic Transport.” *Macromolecular Theory and Simulations* 26.1 (2017), p. 160051.

[7] C. J. Cramer. *Essentials of Computational Chemistry: Theories and Models*. 2nd ed. John Wiley & Sons, Ltd., 2006.

[8] R. A. Friesner. “Ab initio quantum chemistry: Methodology and applications.” *Proceedings of the National Academy of Sciences* 102.19 (2005), pp. 6648–6653.

[9] A. Rahman. “Correlations in the Motion of Atoms in Liquid Argon.” *Phys. Rev.* 136 (2A 1964), A405–A411.

[10] A. Warshel; M. Levitt. “Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme.” *Journal of Molecular Biology* 103.2 (1976), pp. 227–249.

[11] M. B. Liu; G. R. Liu; L. W. Zhou; J. Z. Chang. “Dissipative Particle Dynamics (DPD): An Overview and Recent Developments.” *Archives of Computational Methods in Engineering* 22.4 (2015), pp. 529–556.

[12] P. Español; P. B. Warren. “Perspective: Dissipative particle dynamics.” *The Journal of Chemical Physics* 146.15 (2017), p. 150901.

[13] G. Gompper; T. Ihle; D. M. Kroll; R. G. Winkler. “Multi-Particle Collision Dynamics: A Particle-Based Mesoscale Simulation Approach to the Hydrodynamics of Complex Fluids.” *Advanced Computer Simulation Approaches for Soft Matter Sciences III*. ed. by C. Holm; K. Kremer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–87.

[14] E. De Angelis; M. Chinappi; G. Graziani. “Flow simulations with multi-particle collision dynamics.” *Meccanica* 47.8 (2012), pp. 2069–2077.

[15] J. J. Monaghan. “Smoothed particle hydrodynamics.” *Reports on Progress in Physics* 68.8 (2005), pp. 1703–1759.

[16] M. B. Liu; G. R. Liu. “Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments.” *Archives of Computational Methods in Engineering* 17.1 (2010), pp. 25–76.

[17] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Clarendon Press, 2001.

[18] S. Succi. *The Lattice Boltzmann Equation: For Complex States of Flowing Matter*. Oxford University Press, 2018.

[19] M. Griebel; T. Dornseifer; T. Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction*. Philadelphia, Pa.: SIAM, 1998.

[20] A. Taflove; S. C. Hagness. *Computational electrodynamics: the finite-difference time-domain method*. 3rd ed. Artech House antennas and propagation library. Boston, MA: Artech House, 2005.

[21] R. W. Lewis; P. Nithiarasu; K. N. Seetharamu. *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. Wiley, 2004.

[22] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Berlin: Springer, 2007.

[23] R. Eymard; T. Gallouët; R. Herbin. “Finite volume methods.” *Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. Handbook of Numerical Analysis. Elsevier, 2000.

[24] R. J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge: Cambridge Univ. Press, 2002.

[25] O. P. L. Maître; O. M. Knio. *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*. Scientific Computation. Springer, 2010.

[26] R. M. M. Smeets; U. F. Keyser; D. Krapf; M.-Y. Wu; N. H. Dekker; C. Dekker. “Salt Dependence of Ion Transport and DNA Translocation through Solid-State Nanopores.” *Nano Letters* 6.1 (2006), pp. 89–95.

[27] S. Kesselheim; W. Müller; C. Holm. “Origin of Current Blockades in Nanopore Translocation Experiments.” *Phys. Rev. Lett.* 112 (1 2014), p. 018101.

[28] T. Ertl; M. Krone; S. Kesselheim; K. Scharnowski; G. Reina; C. Holm. “Visual Analysis for Space-Time Aggregation of Biomolecular Simulations.” *Faraday Discussions* 169 (2014), pp. 167–178.

[29] G. Rempfer. “Electrokinetic Transport Phenomena in Soft-Matter Systems.” PhD thesis. University of Stuttgart, 2018.

[30] S. Kesselheim; M. Sega; C. Holm. "Effects of dielectric mismatch and chain flexibility on the translocation barriers of charged macromolecules through solid state nanopores." *Soft Matter* 8.36 (2012), pp. 9480–9486.

[31] H. J. Limbach; A. Arnold; B. A. Mann; C. Holm. "ESPResSo – An Extensible Simulation Package for Research on Soft Matter Systems." *Computer Physics Communications* 174.9 (2006), pp. 704–727.

[32] A. Arnold; O. Lenz; S. Kesselheim; R. Weeber; F. Fahrenberger; D. Roehm; P. Koovan; C. Holm. "ESPResSo 3.1: Molecular Dynamics Software for Coarse-Grained Models." *Meshfree Methods for Partial Differential Equations VI*. ed. by M. Griebel; M. A. Schweitzer. Vol. 89. Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2012, pp. 1–23.

[33] F. Weik; R. Weeber; K. Szuttor; K. Breitsprecher; J. de Graaf; M. Kuron; J. Landsgesell; H. Menke; D. Sean; C. Holm. "ESPResSo 4.0 – an extensible software package for simulating soft matter systems." *The European Physical Journal Special Topics* 227.14 (2019), pp. 1789–1816.

[34] J. D. Halverson; T. Brandes; O. Lenz; A. Arnold; S. Bevc; V. Starchenko; K. Kremer; T. Stuehn; D. Reith. "ESPResSo++: A modern multiscale simulation package for soft matter systems." *Computer Physics Communications* 184.4 (2013), pp. 1129–1149.

[35] H. V. Guzman; N. Tretyakov; H. Kobayashi; A. C. Fogarty; K. Kreis; J. Krajniak; C. Junghans; K. Kremer; T. Stuehn. "ESPResSo++ 2.0: Advanced methods for multiscale molecular simulation." *Computer Physics Communications* 238 (2019), pp. 66–76.

[36] S. Plimpton. "Fast Parallel Algorithms for Short-Range Molecular Dynamics." *Journal of Computational Physics* 117.1 (1995), pp. 1–19.

[37] H. Berendsen; D. van der Spoel; R. van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation." *Computer Physics Communications* 91.1 (1995), pp. 43–56.

[38] M. J. Abraham; T. Murtola; R. Schulz; S. Páll; J. C. Smith; B. Hess; E. Lindahl. "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers." *SoftwareX* 1-2 (2015), pp. 19–25.

[39] M. Buchholz; H.-J. Bungartz; J. Vrabec. "Software design for a highly parallel molecular dynamics simulation framework in chemical engineering." *Journal of Computational Science* 2.2 (2011). Simulation Software for Supercomputers, pp. 124–129.

[40] C. Niethammer; S. Becker; M. Bernreuther; M. Buchholz; W. Eckhardt; A. Heinecke; S. Werth; H.-J. Bungartz; C. W. Glass; H. Hasse; J. Vrabec; M. Horsch. "ls1 mardyn: The Massively Parallel Molecular Dynamics Code for Large Systems." *J. Chem. Theory Comput.* 10.10 (2014), pp. 4455–4464.

[41] S. Seckler; N. Tchipev; H.-J. Bungartz; P. Neumann. "Load Balancing for Molecular Dynamics Simulations on Heterogeneous Architectures." *2016 IEEE 23rd International Conference on High Performance Computing*. 2016, pp. 101–110.

[42] J. Vrabec; M. Bernreuther; H.-J. Bungartz; W.-L. Chen; W. Cordes; R. Fingerhut; C. W. Glass; J. Gmehling; R. Hamburger; M. Heilig; M. Heinen; M. T. Horsch; C.-M. Hsieh; M. Hülsmann; P. Jäger; P. Klein; S. Knauer; T. Köddermann; A. Köster; K. Langenbach; S.-T. Lin; P. Neumann; J. Rarey; D. Reith; G. Rutkai; M. Schappals; M. Schenk; A. Schedemann; M. Schönherr; S. Seckler; S. Stephan; K. Stöbener; N. Tchipev; A. Wafai; S. Werth; H. Hasse.

"SkaSim - Skalierbare HPC-Software für molekulare Simulationen in der chemischen Industrie." *Chemie Ingenieur Technik* 90.3 (2018), pp. 295–306.

[43] N. Tchipev; S. Seckler; M. Heinen; J. Vrabec; F. Gratl; M. Horsch; M. Bernreuther; C. W. Glass; C. Niethammer; N. Hammer; B. Krischok; M. Resch; D. Kranzlmüller; H. Hasse; H.-J. Bungartz; P. Neumann. "TweTriS: Twenty trillion-atom simulation." *The International Journal of High Performance Computing Applications* 0.0 (2019), p. 1094342018819741.

[44] H. Sutter. "The free lunch is over: A fundamental turn toward concurrency in software." *Dr. Dobb's Journal* 30.3 (2005), pp. 202–210.

[45] D. E. Keyes. "Domain decomposition in the mainstream of computational science." *Proceedings of the 14 international conference on Domain Decomposition Methods*. UNAM Press, Mexico City, 2003, pp. 79–93.

[46] *MPI Standard 3.1*. 2015.

[47] NVIDIA. *CUDA RUNTIME API V 10.1.243*. 2019.

[48] M. Müller; T. Aoki. "Hybrid Fortran: High Productivity GPU Porting Framework Applied to Japanese Weather Prediction Model." *Accelerator Programming Using Directives*. Ed. by S. Chandrasekaran; G. Juckeland. Cham: Springer International Publishing, 2018, pp. 20–41.

[49] A. S. Tanenbaum; M. Van Steen. *Distributed Systems: Principles and Paradigms*. 2nd ed. Pearson Prentice Hall, 2007.

[50] F. Weik; S. Kesselheim; C. Holm. "A coarse-grained DNA model for the prediction of current signals in DNA translocation experiments." *The Journal of Chemical Physics* 145.19 (2016), p. 194106.

[51] A. C. Calder; B. C. Curts; L. J. Dursi; B. Fryxell; G. Henry; P. MacNece; K. Olson; P. Ricker; R. Rosner; F. X. Timmes; H. M. Tufo; J. W. Truran; M. Zingale. "High-Performance Reactive Fluid Flow Simulations Using Adaptive Mesh Refinement on Thousands of Processors." *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. 2000, pp. 56–56.

[52] J. Rudi; A. C. I. Malossi; T. Isaac; G. Stadler; M. Gurnis; P. W. J. Staar; Y. Ineichen; C. Bekas; A. Curioni; O. Ghattas. "An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas: ACM, 2015, 5:1–5:12.

[53] C. Godenschwager; F. Schornbaum; M. Bauer; H. Köstler; U. Rüde. "A framework for hybrid parallel flow simulations with a trillion cells in complex geometries." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*. ACM Press, 2013.

[54] F. Schornbaum; U. Rüde. "Massively Parallel Algorithms for the Lattice Boltzmann Method on NonUniform Grids." *SIAM Journal on Scientific Computing* 38.2 (2016), pp. C96–C126.

[55] F. Schornbaum; U. Rüde. "Extreme-Scale Block-Structured Adaptive Mesh Refinement." *SIAM Journal on Scientific Computing* 40.3 (2018), pp. C358–C387.

[56] F. Schornbaum. "Block-Structured Adaptive Mesh Refinement for Simulations on Extreme-Scale Supercomputers." PhD thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2018, p. 152.

[57] C. Rettinger; U. Rüde. “A coupled lattice Boltzmann method and discrete element method for discrete particle simulations of particulate flows.” *Computers & Fluids* 172 (2018), pp. 706–719.

[58] C. Rettinger; U. Rüde. “Dynamic Load Balancing Techniques for Particulate Flow Simulations.” *Computation* 7 (2019).

[59] M. Schönherr; K. Kucher; M. Geier; M. Stiebler; S. Freudiger; M. Krafczyk. “Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs.” *Computers and Mathematics with Applications* 61.12 (2011), pp. 3730–3743.

[60] S. Lenz; M. Schönherr; M. Geier; M. Krafczyk; A. Pasquali; A. Christen; M. Giometto. “Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU.” *Journal of Wind Engineering and Industrial Aerodynamics* 189 (2019), pp. 151–162.

[61] N. Maruyama; T. Aoki; K. Taura; R. Yokota; M. Wahib; M. Matsuda; K. Fukuda; T. Shimokawabe; N. Onodera; M. Müller; S. Iwasaki. “Highly Productive, High-Performance Application Frameworks for Post-Petascale Computing.” *Advanced Software Technologies for Post-Peta Scale Computing: The Japanese Post-Peta CREST Research Project*. Ed. by M. Sato. Singapore: Springer Singapore, 2019, pp. 77–98.

[62] C. Burstedde; J. Holke. “p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement.” *JUQUEEN Extreme Scaling Workshop 2016*. Ed. by D. Brömmel; W. Frings; B. J. N. Wylie. JSC Internal Report FZJ-JSC-IB-2016-01. Jülich Supercomputing Centre, 2016, pp. 49–54.

[63] U.-L. Pen. “A Linear Moving Adaptive Particle-Mesh N-Body Algorithm.” *The Astrophysical Journal Supplement Series* 100 (1995), pp. 269–280.

[64] M. Berger; P. Colella. “Local adaptive mesh refinement for shock hydrodynamics.” *Journal of Computational Physics* 82.1 (1989), pp. 64–84.

[65] S. Adjerid; J. Flaherty. “A Local Refinement Finite-Element Method for Two-Dimensional Parabolic Systems.” *SIAM Journal on Scientific and Statistical Computing* 9.5 (1988), pp. 792–811.

[66] A. Dubey; A. Almgren; J. Bell; M. Berzins; S. Brandt; G. Bryan; P. Colella; D. Graves; M. Lijewski; F. Löffler; B. O’Shea; E. Schnetter; B. Van Straalen; K. Weide. “A Survey of High Level Frameworks in Block-structured Adaptive Mesh Refinement Packages.” *Journal of Parallel and Distributed Computing* 74.12 (2014), pp. 3217–3227.

[67] R. Deiterding. “Block-structured Adaptive Mesh Refinement - Theory, Implementation and Application.” *ESIAM Proceedings*. Ed. by E. Cancès; V. Louvet; M. Massot. Vol. 34. 2011, pp. 97–150.

[68] BoxLib. *https://boxlib-codes.github.io/*. 2011.

[69] A. S. Almgren; J. B. Bell; M. J. Lijewski; Z. Luki; E. Van Andel. “Nyx: A Massively Parallel AMR Code for Computational Cosmology.” *The Astrophysical Journal* 765.1 (2013), p. 39.

[70] W. Zhang; A. Almgren; M. Day; T. Nguyen; J. Shalf; D. Unat. “BoxLib with Tiling: An AMR Software Framework.” *SIAM J. Scientific Computing* (2016).

[71] M. Zingale; A. S. Almgren; M. G. Barrios Sazo; V. E. Beckner; J. B. Bell; B. Friesen; A. M. Jacobs; M. P. Katz; C. M. Malone; A. J. Nonaka; D. E. Willcox; W. Zhang. “Meeting the Challenges of Modeling Astrophysical Thermonuclear Explosions: Castro, Maestro and the AMReX Astrophysics Suite.” 2017.

[72] E. Schnetter; S. H. Hawley; I. Hawke. “Evolutions in 3D numerical relativity using fixed mesh refinement.” *Classical and Quantum Gravity* 21.6 (2004), p. 1465.

[73] E. Schnetter; P. Diener; E. N. Dorband; M. Tiglio. “A multi-block infrastructure for three-dimensional time-dependent numerical relativity.” *Classical and Quantum Gravity* 23.16 (2006), S553.

[74] T. Goodale; G. Allen; G. Lanfermann; J. Massó; T. Radke; E. Seidel; J. Shalf. “The Cactus Framework and Toolkit: Design and Applications.” *Proceedings of the 5th International Conference on High Performance Computing for Computational Science*. VECPAR’02. Porto, Portugal: Springer-Verlag, 2003, pp. 197–227.

[75] M. Adams; P. Colella; D. Graves; J. N. Johnson; N. D. Keen; T. J. Ligocki; D. F. Martin; P. W. McCorquodale; D. Modiano; P. O. Schwartz; T. D. Sternberg; B. Van Straalen. *Chombo Software Package for AMR Applications Design Document*. Tech. rep. Applied Numerical Algorithms Group, Computational Research Division, Larence Berkeley National Laboratory, Berkeley, CA, 2015.

[76] G. L. Bryan; M. L. Norman; B. W. O’Shea; T. Abel; J. H. Wise; M. J. Turk; D. R. Reynolds; D. C. Collins; P. Wang; S. W. Skillman; B. Smith; R. P. Harkness; J. Bordner; J.-h. Kim; M. Kuhlen; H. Xu; N. Goldbaum; C. Hummels; A. G. Kritsuk; E. Tasker; S. Skory; C. M. Simpson; O. Hahn; J. S. Oishi; G. C. So; F. Zhao; R. Cen; Y. Li; T. E. Collaboration. “ENZO: An Adaptive Mesh Refinement Code for Astrophysics.” *The Astrophysical Journal Supplement Series* 211.2 (2014), p. 19.

[77] T. Weinzierl; M. Mehl. “Peano A Traversal and Storage Scheme for Octree-Like Adaptive Cartesian Multiscale Grids.” *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2732–2760.

[78] M. Wahib; N. Maruyama; T. Aoki. “Daino: A High-Level Framework for Parallel and Efficient AMR on GPUs.” *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 621–632.

[79] H. Sundar; R. S. Sampath; G. Biros. “Bottom-Up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel.” *SIAM Journal on Scientific Computing* 30.5 (2008), pp. 2675–2708.

[80] B. Fryxell; K. Olson; P. Ricker; F. X. Timmes; M. Zingale; D. Q. Lamb; P. MacNeice; R. Rosner; J. W. Truran; H. Tufo. “FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes.” *The Astrophysical Journal Supplement Series* 131.1 (2000), p. 273.

[81] A. Dubey; K. Antypas; M. K. Ganapathy; L. B. Reid; K. Riley; D. Sheeler; A. Siegel; K. Weide. “Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code.” *Parallel Computing* 35.10 (2009), pp. 512–522.

[82] H. Schive; Y. Tsai; T. Chiueh. “Gamer: A graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics.” *The Astrophysical Journal Supplement Series* 186 (2010), p. 457.

[83] T. Tu; D. R. O’Hallaron; O. Ghattas. “Scalable Parallel Octree Meshing for TeraScale Applications.” *SC ’05: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE, 2005.

[84] A. Lintermann; S. Schlimpert; J. Grimmen; C. Günther; M. Meinke; W. Schröder. “Massively parallel grid generation on HPC systems.” *Computer Methods in Applied Mechanics and Engineering* 277 (2014), pp. 131–153.

[85] C. Burstedde; L. C. Wilcox; O. Ghattas. “p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees.” *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133.

[86] T. Isaac; C. Burstedde; L. C. Wilcox; O. Ghattas. “Recursive Algorithms for Distributed Forests of Octrees.” *SIAM J. Sci. Comput.* 37.5 (2015), pp. C497–C531.

[87] J. Holke. “Scalable Algorithms for Parallel Tree-based Adaptive Mesh Refinement with General Element Types.” PhD thesis. Universität Bonn, 2018.

[88] C. Burstedde; J. Holke. “A Tetrahedral Space-Filling Curve for Nonconforming Adaptive Meshes.” *SIAM Journal on Scientific Computing* 38.5 (2016), pp. C471–C503.

[89] H. G. Klimach; M. Hasert; J. Zudrop; S. P. Roller. “Distributed octree mesh infrastructure for flow simulations.” *European Congress on Computational Methods in Applied Sciences and Engineering*. 2012, pp. 1–15.

[90] S. Roller; J. Bernsdorf; H. Klimach; M. Hasert; D. Harlacher; M. Cakircali; S. Zimny; K. Masilamani; L. Didinger; J. Zudrop. “An Adaptable Simulation Framework Based on a Linearized Octree.” *High Performance Computing on Vector Systems 2011*. Ed. by M. Resch; X. Wang; W. Bez; E. Focht; H. Kobayashi; S. Roller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 93–105.

[91] H. Klimach; K. Jain; S. Roller. “End-to-end parallel simulations with APES.” *Advances in Parallel Computing* 25 (2014), pp. 703–711.

[92] S. G. Parker. “A component-based architecture for parallel multi-physics PDE simulation.” *Future Generation Computer Systems* 22.1-2 (2006), pp. 204–216.

[93] C. Feichtinger; S. Donath; H. Köstler; J. Götz; U. Rüde. “WaLBerla: HPC software design for computational engineering simulations.” *Journal of Computational Science* 2 (2 2011), pp. 105–112.

[94] G. M. Morton. *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*. Tech. rep. IBM Ltd., 1966.

[95] C. Burstedde; J. Holke; T. Isaac. “Bounds on the number of discontinuities of Morton-type space-filling curves” (2015).

[96] D. Hilbert. “Über die stetige Abbildung einer Linie auf ein Flächenstück.” *Mathematische Annalen* 38 (1891), pp. 459–460.

[97] G. Peano. “Sur une courbe, qui remplit toute une aire plane.” *Mathematische Annalen* 36.1 (1890), pp. 157–160.

[98] W. Sierpiski. “Sur une nouvelle courbe continue qui remplit toute une aire plane.” *Bull. Acad. Sci. Cracovie (Sci. math. et nat. Serie A)* (1912), pp. 462–478.

[99] M. Bader. *Space-filling curves: an introduction with applications in scientific computing*. Heidelberg, New York [u.a.]: Springer, 2013.

[100] P. M. Campbell; K. D. Devine; J. E. Flaherty; L. G. Gervasio; J. D. Teresco. “Dynamic octree load balancing using space-filling curves.” *Williams College Department of Computer Science, Technical Report* (2003), pp. 1–26.

[101] W. F. Mitchell. “The full domain partition approach to distributing adaptive grids.” *Applied Numerical Mathematics* 26.1 (1998), pp. 265–275.

[102] D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[103] J. M. Morris. “Traversing binary trees simply and cheaply.” *Information Processing Letters* 9.5 (1979), pp. 197–200.

[104] A. Pnar; C. Aykanat. “Fast optimal load balancing algorithms for 1D partitioning.” *Journal of Parallel and Distributed Computing* 64.8 (2004), pp. 974–996.

[105] T. Tu; D. R. O’Hallaron. “Balance refinement of massive linear octree datasets.” *Technical Report CMU-CS-04* (2004).

[106] F. Kohlgrüber. “Visualisierung von oktalbaumbasierten kartesischen Gittern.” Bachelor’s thesis. Universität Stuttgart, IPVS/SGS, 2016.

[107] T. Isaac; C. Burstedde; O. Ghattas. “Low-cost parallel algorithms for 2:1 octree balance.” *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012* (2012), pp. 426–437.

[108] L. Greengard; V. Rokhlin. “A fast algorithm for particle simulations.” *Journal of Computational Physics* 73.2 (1987), pp. 325–348.

[109] M. S. Warren; J. K. Salmon. “A Parallel Hashed Oct-Tree N-body Algorithm.” *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*. Supercomputing ’93. Portland, Oregon, USA: ACM, 1993, pp. 12–21.

[110] L. Dalcin; R. Bradshaw; K. Smith; C. Citro; S. Behnel; D. S. Seljebotn. “Cython: The Best of Both Worlds.” *Computing in Science & Engineering* 13 (2010), pp. 31–39.

[111] G. Inci; A. Arnold; A. Kronenburg; R. Weeber. “Modeling Nanoparticle Agglomeration using Local Interactions.” *Aerosol Science and Technology* 48.8 (2014), pp. 842–852.

[112] C. Schober; D. Keerl; M. Lehmann; M. Mehl. “Simulating The Interaction of Electrostatically Charged Particles in the Inflow Area of Cabin Air Filters Using a Fully Coupled System.” *Coupled Problems in Science and Engineering VII*. ed. by M. Papadrakakis; E. Oñate; B. Schrefler. 2017.

[113] J. Höpfner; T. Richter; P. Koovan; C. Holm; M. Wilhelm. “Seawater desalination via hydrogels: Coarse grained simulations and practical realisation.” *Progr. Colloid. Polym. Sci.* Progress in Colloid and Polymer Science 140.140 (2013). Ed. by G. Sadowski; W. Richtering.

[114] B. J. Reynwar; G. Illya; V. A. Harmandaris; M. M. Muller; K. Kremer; M. Deserno. “Aggregation and vesiculation of membrane proteins by curvature-mediated interactions.” *Nature* 447.7143 (2007), pp. 461–464.

[115] M. Kuron; A. Arnold. “Role of geometrical shape in like-charge attraction of DNA.” *European Physical Journal E: Soft Matter* 38 (2015), p. 20.

[116] K. Breitsprecher; P. Koovan; C. Holm. “Coarse-grained simulations of an ionic liquid-based capacitor: I. Density, ion size, and valency effects.” *Journal of Physics: Condensed Matter* 26.28 (2014), p. 284108.

[117] K. Breitsprecher; P. Koovan; C. Holm. “Coarse-grained simulations of an ionic liquid-based capacitor: II. Asymmetry in ion shape and charge localization.” *Journal of Physics: Condensed Matter* 26.28 (2014), p. 284114.

[118] F. Capuani; I. Pagonabarraga; D. Frenkel. “Discrete solution of the electrokinetic equations.” *The Journal of Chemical Physics* 121.2 (2004), pp. 973–986.

[119] A. Einstein. “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen.” *Annalen der Physik* 322.8 (1905), pp. 549–560.

[120] M. von Smoluchowski. “Zur kinetischen Theorie der Brownschen Molekularbewegung und der Suspensionen.” *Annalen der Physik* 326.14 (1906), pp. 756–780.

[121] G. Rempfer; G. B. Davies; C. Holm; J. de Graaf. “Reducing spurious flow in simulations of electrokinetic phenomena.” *The Journal of Chemical Physics* 145.4 (2016), p. 044901.

[122] Y. H. Qian; D. D’Humières; P. Lallemand. “Lattice BGK Models for Navier-Stokes Equation.” *EPL (Europhysics Letters)* 17.6 (1992), p. 479.

[123] S. Chapman; T. Cowling. “The Mathematical Theory of Non-uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases. Cambridge Mathematical Library” (1970), pp. 27–52.

[124] P. L. Bhatnagar; E. P. Gross; M. Krook. “A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems.” *Physical Review* 94.3 (1954), pp. 511–525.

[125] I. Ginzburg; F. Verhaeghe; D. dHumières. “Two-relaxation-time lattice Boltzmann scheme: About parametrization, velocity, pressure and mixed boundary conditions.” *Communications in computational physics* 3.2 (2008), pp. 427–478.

[126] I. Ginzburg; F. Verhaeghe; D. dHumieres. “Study of simple hydrodynamic solutions with the two-relaxation-times lattice Boltzmann scheme.” *Communications in computational physics* 3.3 (2008), pp. 519–581.

[127] D. dHumières; I. Ginzburg; M. Krafczyk; P. Lallemand; L.-S. Luo. “Multiple-relaxation-time lattice Boltzmann models in three dimensions.” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 360.1792 (2002), pp. 437–451.

[128] L.-S. Luo; W. Liao; X. Chen; Y. Peng; W. Zhang. “Numerics of the lattice Boltzmann method: Effects of collision models on the lattice Boltzmann simulations.” *Physical Review E* 83.5 (2011), p. 056710.

[129] U. D. Schiller. “Thermal fluctuations and boundary conditions in the lattice Boltzmann method.” PhD thesis. Johannes Gutenberg-Universität, Mainz, 2008.

[130] L. Li; R. Mei; J. F. Klausner. “Boundary conditions for thermal lattice Boltzmann equation method.” *Journal of Computational Physics* 237 (2013), pp. 366–395.

[131] R. Deiterding; S. L. Wood. “Predictive wind turbine simulation with an adaptive lattice Boltzmann method for moving boundaries.” *Journal of Physics: Conference Series* 753.8 (2016), p. 082005.

[132] A. Pasquali; M. Geier; M. Krafczyk. “Near-wall treatment for the simulation of turbulent flow by the cumulant lattice Boltzmann method.” *Computers & Mathematics with Applications* (2017).

[133] X. Shan; X.-F. Yuan; H. Chen. “Kinetic Theory Representation of Hydrodynamics: A Way Beyond the NavierStokes Equation.” *Journal of Fluid Mechanics* 550 (2006), pp. 413–441.

[134] M. Geier; A. Greiner; J. G. Korvink. “Cascaded digital lattice Boltzmann automata for high Reynolds number flow.” *Phys. Rev. E* 73 (6 2006), p. 066705.

[135] P. Asinari. “Generalized local equilibrium in the cascaded lattice Boltzmann method.” *Phys. Rev. E* 78 (1 2008), p. 016701.

[136] S. Seeger; H. Hoffmann. “The cumulant method for computational kinetic theory.” *Continuum Mechanics and Thermodynamics* 12.6 (2000), pp. 403–421.

[137] M. Geier; M. Schönherr; A. Pasquali; M. Krafczyk. "The cumulant lattice Boltzmann equation in three dimensions: Theory and validation." *Computers* & *Mathematics with Applications* 70.4 (2015), pp. 507–547.

[138] K. Kutscher; M. Geier; M. Krafczyk. "Multiscale simulation of turbulent flow interacting with porous media based on a massively parallel implementation of the cumulant lattice Boltzmann method." *Computers* & *Fluids* (2018).

[139] M. Wittmann; T. Zeiser; G. Hager; G. Wellein. "Modeling and analyzing performance for highly optimized propagation steps of the lattice Boltzmann method on sparse lattices." *CoRR* abs/1410.0412 (2014).

[140] M. Geier; M. Schönherr. "Esoteric Twist: An Efficient in-Place Streaming Algorithmus for the Lattice Boltzmann Method on Massively Parallel Hardware." *Computation* 5.19 (2017), p. 15.

[141] Adhikari, R.; Stratford, K.; Cates, M. E.; Wagner, A. J. "Fluctuating lattice Boltzmann." *Europhys. Lett.* 71.3 (2005), pp. 473–479.

[142] G. Wellein; T. Zeiser; G. Hager; S. Donath. "On the single processor performance of simple lattice Boltzmann kernels." *Computers & Fluids* 35.8-9 (2006), pp. 910–919.

[143] D. Roehm; A. Arnold. "Lattice Boltzmann simulations on GPUs with ESPResSo." *The European Physical Journal Special Topics* 210.1 (2012), pp. 89–100.

[144] A. Arnold; F. Fahrenberger; C. Holm; O. Lenz; M. Bolten; H. Dachsel; R. Halver; I. Kabadshow; F. Gähler; F. Heber; J. Iseringhausen; M. Hofmann; M. Pippig; D. Potts; G. Sutmann. "Comparison of scalable fast methods for long-range interactions." *Phys. Rev. E* 88 (6 2013), p. 063308.

[145] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1995.

[146] J. M. Haile. *Molecular Dynamics Simulation*. John Wiley & Sons, Ltd., 1997. 512 pp.

[147] M. Griebel; S. Knapek; G. Zumbusch. *Numerical Simulation in Molecular Dynamics*. 2007.

[148] R. W. Hockney; J. W. Eastwood. *Computer Simulation Using Particles*. Bristol, PA, USA: Taylor & Francis, Inc., 1988.

[149] M. Allen; D. Tildesley. *Computer Simulation of Liquids*. Oxford Science Publ. Clarendon Press, 1989.

[150] L. Verlet. "Computer "Experiments"on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules." *Phys. Rev.* 159 (1 1967), pp. 98–103.

[151] J. E. Jones. "On the Determination of Molecular Fields. II. From the Equation of State of a Gas." *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 106.738 (1924), pp. 463–477.

[152] J. E. Lennard-Jones. "Cohesion." *Proceedings of the Physical Society* 43.5 (1931), pp. 461–482.

[153] G. Mie. "Zur kinetischen Theorie der einatomigen Körper." *Annalen der Physik* 316.8 (1903), pp. 657–697.

[154] P. P. Ewald. "Die Berechnung optischer und elektrostatischer Gitterpotentiale." *Annalen der Physik* 369.3 (1921), pp. 253–287.

[155] S. W. de Leeuw; J. W. Perram; E. R. Smith. “Simulation of electrostatic systems in periodic boundary conditions. I. Lattice sums and dielectric constants.” *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 373.1752 (1980), pp. 27–56.

[156] B. Dünweg; A. J. C. Ladd. “Lattice Boltzmann Simulations of Soft Matter Systems” (2009), pp. 89–166.

[157] P. Ahlrichs; B. Dünweg. “Simulation of a single polymer chain in solution by combining lattice Boltzmann and molecular dynamics.” *The Journal of chemical physics* 111.17 (1999), pp. 8225–8239.

[158] O. Filippova; D. Hänel. “Grid Refinement for Lattice-BGK Models.” *Journal of Computational Physics* 147.1 (1998), pp. 219–228.

[159] O. Filippova; D. Hänel. “Boundary-Fitting and Local Grid Refinement for Lattice-BGK Models.” *International Journal of Modern Physics C* 09.08 (1998), pp. 1271–1279.

[160] O. Filippova; D. Hänel. “Acceleration of Lattice-BGK Schemes with Grid Refinement.” *Journal of Computational Physics* 165.2 (2000), pp. 407–427.

[161] D. Yu; R. Mei; W. Shyy. “A multi-block lattice Boltzmann method for viscous fluid flows.” *International Journal for Numerical Methods in Fluids* 39.2 (), pp. 99–120.

[162] B. Crouse; E. Rank; M. Krafczyk; J. Tölke. “A LB-Based Approach For Adaptive Flow Simulations.” *International Journal of Modern Physics B* 17.01n02 (2003), pp. 109–112.

[163] A. Dupuis; B. Chopard. “Theory and applications of an alternative lattice Boltzmann grid refinement algorithm.” *Phys. Rev. E* 67 (6 2003), p. 066707.

[164] J. Tölke; S. Freudiger; M. Krafczyk. “An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations.” *Computers & Fluids* 35.8 (2006). Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science, pp. 820–830.

[165] D. Lagrava; O. Malaspinas; J. Latt; B. Chopard. “Advances in multi-domain lattice Boltzmann grid refinement.” *Journal of Computational Physics* 231.14 (2012), pp. 4808–4822.

[166] Y. Kuwata; K. Suga. “Imbalance-correction grid-refinement method for lattice Boltzmann flow simulations.” *Journal of Computational Physics* 311 (2016), pp. 348–362.

[167] C. E. Shannon. “Communication in the presence of noise.” *Proceedings of the IRE* 37.1 (1949), pp. 10–21.

[168] H. Chen. “Volumetric formulation of the lattice Boltzmann method for fluid dynamics: Basic concept.” *Physical Review E* 58.3 (1998), pp. 3955–3963.

[169] H. Chen; O. Filippova; J. Hoch; K. Molvig; R. Shock; C. Teixeira; R. Zhang. “Grid refinement in lattice Boltzmann methods based on volumetric formulation.” *Physica A: Statistical Mechanics and its Applications* 362.1 (2006), pp. 158–167.

[170] M. Rohde; D. Kandhai; J. J. Derksen; H. E. A. Van den Akker. “A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes.” *International Journal for Numerical Methods in Fluids* 51.4 (2006), pp. 439–468.

[171] P. Neumann; T. Neckel. “A dynamic mesh refinement technique for Lattice Boltzmann simulations on octree-like grids.” *Computational Mechanics* 51.2 (2012), pp. 237–253.

[172] M. Geier; A. Greiner; J. G. Korvink. “Bubble functions for the lattice Boltzmann method and their application to grid refinement.” *The European Physical Journal Special Topics* 171.1 (2009), pp. 173–179.

[173] J. Tölke; M. Krafczyk. “Second order interpolation of the flow field in the lattice Boltzmann method.” *Computers & Mathematics with Applications* 58.5 (2009). Mesoscopic Methods in Engineering and Science, pp. 898–902.

[174] A. Fakhari; T. Lee. “Finite-difference lattice Boltzmann method with a block-structured adaptive-mesh-refinement technique.” *Physical Review E* 89.3 (2014), p. 033310.

[175] A. Fakhari; T. Lee. “Numerics of the lattice boltzmann method on nonuniform grids: Standard LBM and finite-difference LBM.” *Computers & Fluids* 107 (2015), pp. 205–213.

[176] A. Fakhari; M. Geier; T. Lee. “A mass-conserving lattice Boltzmann method with dynamic grid refinement for immiscible two-phase flows.” *Journal of Computational Physics* 315 (2016), pp. 434–457.

[177] X. Shi; J. Lin; Z. Yu. “Discontinuous Galerkin spectral element lattice Boltzmann method on triangular element.” *International Journal for Numerical Methods in Fluids* 42.11 (2003), pp. 1249–1261.

[178] M. Min; T. Lee. “A spectral-element discontinuous Galerkin lattice Boltzmann method for nearly incompressible flows.” *Journal of Computational Physics* 230.1 (2011), pp. 245–259.

[179] M. D. Mazzeo. “Fast discontinuous Galerkin lattice-Boltzmann simulations on GPUs via maximal kernel fusion.” *Computer Physics Communications* 184.3 (2013), pp. 537–549.

[180] K. C. Uga; M. Min; T. Lee; P. F. Fischer. “Spectral-element discontinuous Galerkin lattice Boltzmann simulation of flow past two cylinders in tandem with an exponential time integrator.” *Computers & Mathematics with Applications* 65.2 (2013). Special Issue on Mesoscopic Methods in Engineering and Science (ICMMES-2010, Edmonton, Canada), pp. 239–251.

[181] S. S. Patel; M. Min; K. C. Uga; T. Lee. “A spectral-element discontinuous Galerkin lattice Boltzmann method for simulating natural convection heat transfer in a horizontal concentric annulus.” *Computers & Fluids* 95 (2014), pp. 197–209.

[182] A. Zadehgol; M. Ashrafizaadeh; S. Musavi. “A nodal discontinuous Galerkin lattice Boltzmann method for fluid flow problems.” *Computers & Fluids* 105 (2014), pp. 58–65.

[183] S.-L. Han; P. Zhu; Z.-Q. Lin. “Two-dimensional interpolation-supplemented and Taylor-series expansion-based lattice Boltzmann method and its application.” *Communications in Nonlinear Science and Numerical Simulation* 12.7 (2007), pp. 1162–1171.

[184] K. Qu; C. Shu; Y. T. Chew. “An Isoparametric Transformation-Based Interpolation-Supplemented Lattice Boltzmann Method and Its Application.” *Modern Physics Letters B* 24.13 (2010), pp. 1315–1318.

[185] M. Mirzaei; A. Poozesh. “Simulation of fluid flow in a body-fitted grid system using the lattice Boltzmann method.” *Phys. Rev. E* 87 (6 2013), p. 063312.

[186] Y. Li; E. J. LeBoeuf; P. K. Basu. “Least-squares finite-element scheme for the lattice Boltzmann method on an unstructured mesh.” *Phys. Rev. E* 72 (4 2005), p. 046711.

[187] D. V. Patil; K. Lakshmisha. “Finite volume TVD formulation of lattice Boltzmann simulation on unstructured mesh.” *Journal of Computational Physics* 228.14 (2009), pp. 5262–5279.

[188] T. Hübner; S. Turek. “Efficient monolithic simulation techniques for the stationary Lattice Boltzmann equation on general meshes.” *Computing and Visualization in Science* 13.3 (2010), pp. 129–143.

[189] A. Harten. “Multiresolution algorithms for the numerical solution of hyperbolic conservation laws.” *Communications on Pure and Applied Mathematics* 48.12 (1995), pp. 1305–1342.

[190] D. L. George; R. J. LeVeque. “Finite Volume Methods and Adaptive Refinement for Global Tsunami Propagation and Local Inundation.” *Science of Tsunami Hazards* 24.5 (2006), pp. 319–328.

[191] C. Burstedde; D. Calhoun; K. Mandli; A. R. Terrel. “ForestClaw: Hybrid forest-of-octrees AMR for hyperbolic conservation laws.” *Advances in Parallel Computing* 25 (2014), pp. 253–262.

[192] D. A. Calhoun; C. Burstedde. “ForestClaw: A parallel algorithm for patch-based adaptive mesh refinement on a forest of quadtrees.” *CoRR* abs/1703.03116 (2017).

[193] F. Losasso; F. Gibou; R. Fedkiw. “Simulating Water and Smoke with an Octree Data Structure.” *ACM SIGGRAPH 2004 Papers*. SIGGRAPH ’04. Los Angeles, California: ACM, 2004, pp. 457–462.

[194] F. Losasso; R. Fedkiw; S. Osher. “Spatially adaptive techniques for level set methods and incompressible flow.” *Computers & Fluids* 35.10 (2006), pp. 995–1010.

[195] C. Min; F. Gibou; H. D. Ceniceros. “A supra-convergent finite difference scheme for the variable coefficient Poisson equation on non-graded grids.” *Journal of Computational Physics* 218.1 (2006), pp. 123–140.

[196] H. Chen; C. Min; F. Gibou. “A Supra-Convergent Finite Difference Scheme for the Poisson and Heat Equations on Irregular Domains and Non-Graded Adaptive Cartesian Grids.” *J. Sci. Comput.* 31.1-2 (2007), pp. 19–60.

[197] M. A. Olshanskii; K. M. Terekhov; Y. V. Vassilevski. “An octree-based solver for the incompressible Navier–Stokes equations with enhanced stability and low dissipation.” *Computer & Fluids* 84 (2013), pp. 231–246.

[198] C. Xiaolin; M. Zeyao. “A New Scalable Parallel Method for Molecular Dynamics Based on Cell-Block Data Structure.” *Parallel and Distributed Processing and Applications*. Ed. by J. Cao; L. Yang; M. Guo; F. Lau. Vol. 3358. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 757–764.

[199] J. A. Anderson; C. D. Lorenz; A. Travesset. “General purpose molecular dynamics simulations fully implemented on graphics processing units.” *Journal of Computational Physics* 227.10 (2008), pp. 5342–5359.

[200] B. Hendrickson; K. Devine. “Dynamic load balancing in computational mechanics.” *Computer Methods in Applied Mechanics and Engineering* 184.2 (2000), pp. 485–500.

[201] S. Tirthapura; S. Seal; S. Aluru. “A Formal Analysis of Space Filling Curves for Parallel Domain Decomposition.” *2006 International Conference on Parallel Processing (ICPP’06)*. 2006, pp. 505–512.

[202] A. Nakano; T. Campbell. “An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multiresolution molecular dynamics.” *Parallel Computing* 23.10 (1997), pp. 1461–1478.

[203] M. Buchholz. “Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen.” PhD thesis. München: Institut für Informatik, Technische Universität München, 2010.

[204] R. Prat; L. Colombet; R. Namyst. “Combining Task-based Parallelism and Adaptive Mesh Refinement Techniques in Molecular Dynamics Simulations.” *Proceedings of the 47th International Conference on Parallel Processing*. ICPP 2018. Eugene, OR, USA: ACM, 2018, 48:1–48:10.

[205] S. Hirschmann; M. Brunn; M. Lahnert; C. W. Glass; M. Mehl; D. Pflüger. “Load Balancing with p4est for Short-Range Molecular Dynamics with ESPResSo.” Ed. by S. Bassini; M. Danelutto; P. Dazzi; G. R. Joubert; F. Peters. Vol. 32. Advances in Parallel Computing. IOS Press, 2017, pp. 455–464.

[206] M. Mehl; M. Lahnert. “Adaptive grid implementation for parallel continuum mechanics methods in particle simulations.” *The European Physical Journal Special Topics* 227.14 (2019), pp. 1757–1778. Erratum. Lahnert; Burstedde; Mehl [226].

[207] S. Hirschmann; M. Lahnert; C. Schober; M. Brunn; M. Mehl; D. Pflüger. “Load-Balancing and Spatial Adaptivity for Coarse-Grained Molecular Dynamics Applications.” *High Performance Computing in Science and Engineering ’18*. Ed. by W. E. Nagel; D. H. Kröner; M. M. Resch. Springer International Publishing, 2018, pp. 409–423.

[208] S. Hirschmann; D. Pflüger; C. W. Glass. “Towards Understanding Optimal Load-Balancing of Heterogeneous Short-Range Molecular Dynamics.” *Workshop on High Performance Computing and Big Data in Molecular Engineering 2016 (HBME 2016)*. Hyderabad, India, 2016.

[209] C. Burstedde; O. Ghattas; G. Stadler; T. Tu; L. C. Wilcox. “Towards adaptive mesh PDE simulations on petascale computers.” *Proceedings of Teragrid* 8 (2008).

[210] L. C. Wilcox; G. Stadler; C. Burstedde; O. Ghattas. “A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media.” *Journal of Computational Physics* 229.24 (2010), pp. 9373–9396.

[211] C. Burstedde. *Parallel tree algorithms for AMR and non-standard data access*. 2018.

[212] O. A. Hickey; C. Holm; J. L. Harden; G. W. Slater. “Implicit method for simulating electrohydrodynamics of polyelectrolytes.” *Physical review letters* 105.14 (2010), p. 148301.

[213] G. Rempfer. “A Lattice based Model for Electrokinetics.” Master’s thesis. University of Stuttgart, 2013.

[214] G. M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.” *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS ’67 (Spring). Atlantic City, New Jersey: ACM, 1967, pp. 483–485.

[215] J. L. Gustafson. “Reevaluating Amdahl’s Law.” *Commun. ACM* 31.5 (1988), pp. 532–533.

[216] S. Hirschmann; C. W. Glass; D. Pflüger. “Enabling unstructured domain decompositions for inhomogeneous short-range molecular dynamics in ESPResSo.” *The European Physical Journal Special Topics* 227.14 (2019), pp. 1779–1788.

[217] G. Karypis; V. Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.” *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392.

[218] K. Schloegel; G. Karypis; V. Kumar. “Parallel Multilevel Algorithms for Multi-constraint Graph Partitioning (Distinguished Paper).” *Proceedings from the 6th International Euro-Par*

Conference on Parallel Processing. Euro-Par ’00. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 296–310.

[219] C. Begau; G. Sutmann. “Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations.” *Computer Physics Communications* 190.0 (2015), pp. 51–61.

[220] W. Hackbusch. *Multi-grid methods and applications*. Vol. 4. Springer series in computational mathematics. Berlin [u.a.]: Springer, 1985, pp. XIV, 377.

[221] H. Sundar; G. Biros; C. Burstedde; J. Rudi; O. Ghattas; G. Stadler. “Parallel geometric-algebraic multigrid on unstructured forests of octrees.” *SC12: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2012.

[222] B. Kurz. “Lattice-Boltzmann Simulationen auf mehreren GPUs.” Bachelor’s thesis. University of Stuttgart, 2018.

[223] P. Bastian; M. Blatt; A. Dedner; C. Engwer; R. Klöfkorn; M. Ohlberger; O. Sander. “A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework.” *Computing* 82.2 (2008), pp. 103–119.

[224] P. Bastian; M. Blatt; A. Dedner; C. Engwer; R. Klöfkorn; R. Kornhuber; M. Ohlberger; O. Sander. “A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE.” *Computing* 82.2 (2008), pp. 121–138.

[225] T. Weinzierl; R. Wittmann; K. Unterweger; M. Bader; A. Breuer; S. Rettenberger. “Hardware-aware block size tailoring on adaptive spacetree grids for shallow water waves.” *HiStencils 2014 - Proceedings of the 1st international workshop on high-performance stencil computations.* Ed. by A. GröSSlinger; H. Köstler. HiPEAC. HiStencils, 2014, pp. 57–64.

[226] M. Lahnert; C. Burstedde; M. Mehl. “Erratum to: Adaptive grid implementation for parallel continuum mechanics methods in particle simulations.” *European Physical Journal Special Topics* 227 (2019), pp. 1757–1778.

All URLs have last been checked on August 28, 2019.

C Declaration of Authorship

I, Michael Stefan Lahnert, declare that this thesis titled, “Adaptive Grid Implementation for Parallel Continuum Mechanics Methods in Particle Simulations” and the work presented in it, are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Stuttgart.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Stuttgart, August 28, 2019

Michael Stefan Lahnert