Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 36

Coupling of Particle Simulation and Lattice Boltzmann Background Flow on Adaptive Grids

Malte Brunn

Course of Study:	Simulation Technology
Examiner:	Prof. Dr. Miriam Mehl
Supervisor:	Steffen Hirschmann, Michael Lahnert
Commenced:	December 04, 2016
Completed:	June 02, 2017
CR-Classification:	G1, G1.7, G1.8, G4

"We are stuck with technology when what we really want is just stuff that works."

Douglas Adams

Abstract

The lattice-Boltzmann method as well as classical molecular dynamics are established and widely used methods for the simulation and research of soft matter. Molecular dynamics is a computer simulation technique on microscopic scales solving the multi-body kinetic equations of the involved particles. The lattice-Boltzmann method describes the hydrodynamic interactions of fluids, gases, or other soft matter on a coarser scale. Many applications, however, are multi-scale problems and they require a coupling of both methods. A basic concept for short ranged interactions in molecular dynamics is the linked cells algorithm that is in $\mathcal{O}(N)$ for homogeneously distributed particles. Spatial adaptive methods for the lattice-Boltzmann scheme are used in order to reduce costly scaling effects on the runtime and memory for large-scale simulations. As basis for this work the highly flexible simulation software ESPRESSO is used and extended. The adaptive lattice-Boltzmann scheme, that is implemented in ESPRESSO, uses a domain decomposition with tree-based grids along the space-filling Morton curve using the P4EST software library. However, coupling the regular particle simulation with the adaptive lattice-Boltzmann method for highly parallel computer architectures is a challenging issue that raises several problems.

In this work, an approach for the domain decomposition of the linked cells algorithm based on space-filling curves and the P4EST library is presented. In general, the grids for molecular dynamics and fluid simulations are not equal. Thus, strategies to distribute differently refined grids on parallel processes are explained, including a parallel algorithm to construct the finest common tree using P4EST. Furthermore, a method for interpolation and extrapolation, that is needed for the viscous coupling of particles with the fluid, on adaptively refined grids are discussed. The ESPRESSO simulation software is augmented by the developed methods for particle-fluid coupling as well as the Morton curve based domain decompositions in a minimally invasive manner. The original ESPRESSO implementation for regular particle and fluid simulations is used as reference for the developed algorithms.

Contents

List of Figures

List of Algorithms

CHAPTER 1

Introduction

A lot of scientific computer simulations, e.g. filters for nanoparticles based on microfibers [WIS10, LWKH16, CKW+13], take place on a wide range of scales. Particles transported by a flow through the filter interact with themselves, the fluid, and the micro fibers. The interactions are considered on the large scale setting of the whole systems. Both, particles and fibers, are very small and interact on a intermolecular space-time scale. The whole filter structure, however, is much larger with respect to relevant time and space scales. Typically, these multi-scale problems are computational very complex and time-consuming. The regular structure of numerical discretization schemes, as for instance the lattice-Boltzmann method for fluid dynamics, are a big issue that limits the simulation in terms of runtime and size, due to the algorithm's complexity and restricted hardware resources. Basic concepts to reduce the computational workload are methods for spatial adaptivity. Areas of interest, e.g. the surrounding of the micro fibers, are locally refined while the plain fluid in between is considered on a coarser scale. However, adaptivity usually comes along with more sophisticated algorithms and parallelization schemes. Coupling particles with a refined grid, in particular, raises new problems. Interpolation on the adaptive grid as well as parallelization strategies for the used algorithms become more involved. Thus, this work is dedicated to resolving those issues of the particle-fluid coupling in order to benefit from adaptivity.

In Chapter 2 the used software packages, ESPRESSO [LAMH06, ALK+ed] and P4EST [BWG11], are presented. The ESPRESSO simulation software is extended in the scope of this work to enable the viscous fluid-particle coupling for the adaptive lattice-Boltzmann algorithm. The P4EST library is used to manage tree-based grids and the parallel domain decomposition. Some basic algorithms needed for the work with P4EST that operates on the space-filling Morton curve are presented in this chapter as well.

Chapter 3 covers the Boltzmann equation and its discretization the lattice-Boltzmann method. Starting with an overview of the formal derivation for the Boltzmann equation from the kinetic multi-body system, the regular discretization of the phase-time space based on the D3Q19 model is explained. This discretization results in the lattice-Boltzmann scheme. Moreover, the spatial adaptive lattice-Boltzmann method proposed by Rohde et al. [RKDA06] and its implementation in ESPRESSO by Lahnert et al. [LBH+16] is outlined. Tölke et al. [TFK06] presented a similar approach for adaptive lattice-Boltzmann fluids on tree-structured grids based on the two-phase flow with a boundary interface.

In Chapter 4 the general molecular dynamic simulation for short ranged interaction using the linked cells scheme is explained. A common parallelization approach based on the spatial domain decomposition with cubic domains is introduced. Buchholz [Buc10] discusses several strategies for load balancing of parallel molecular dynamic simulations including the spatial decomposition along space-filling curves. The irregular domain decomposition of the grid resulting from the Morton curve used in P4EST, however, raises new problems for the integration in the ESPRESSO software that are discussed and resolved. Moreover, the developed methods for parallel molecular dynamics on regular tree-structured grids using the P4EST library for domain decomposition are presented.

The viscous fluid-particle coupling for the P4EST based molecular dynamics with the adaptive lattice-Boltzmann method is presented in Chapter 5. The coupling is based on the scheme for regular grids presented by Ahlrichs and Dünweg [AD99]. For an irregular decomposition of similar fluid and particle grids a repartitioning method based on the finest common tree is discussed. Spatial decompositions of grids with arbitrary mesh widths, however, have overlapping grid cells intersecting the process boundary. A decomposition approach of the linked cells grid is proposed for those cases as well.

The results of this work are reviewed in Chapter 6 including a runtime comparison of the regular molecular dynamics with the developed P4EST based particle simulation and some basic tests for coupling with regular lattice-Boltzmann grids using the adaptive implementation.

CHAPTER 2

Basic Software Packages

In this chapter the underlying software packages for this work are introduced and some basic features are discussed. The simulation software ESPRESSO is extended by the algorithms developed in the scope of this work. The adaptive structures and the irregular domain decomposition along space-filling curves are based on the P4EST library.

2.1 The Simulation Software ESPRESSO

ESPRESSO (**E**xtensible **S**imulation **P**ackage for **Res**earch on **So**ft Matter Systems) [LAMH06, ALK+ed] is an open-source software package designed to simulate and investigate soft matter systems with molecular dynamics and *Monte Carlo* simulations. The term *soft matter* is used for materials that are neither regular fluids nor solid objects, as for example polymers, colloids, liquid crystals, glasses, or dipolar fluids. The package includes a wide range of advanced algorithms for statistical ensembles, bonded and non-bonded potentials, anisotropic interactions as well as methods for electrostatics, dielectrics, magnetostatics, and hydrodynamics. In addition to the molecular dynamic simulation, ESPRESSO is also capable of solving the fluctuating Navier-Stokes equation with the lattice-Boltzmann method [Sch08]. Lahnert et al. [LBH+16] extended the implemented lattice-Boltzmann scheme to handle the spatial adaptivity of tree-based grids. The adaptive lattice-Boltzmann method follows the method proposed by Rohde et al. [RKDA06] and it is further discussed in Chapter 3.2. The lattice-Boltzmann fluid simulation can be coupled with molecular dynamics based on the method introduced by Dünweg and Ladd [DL09]. ESPRESSO is used in a variety of applications including, amongst others, nanoparticle agglomeration [IAKW14], the like-charge attraction of DNA chains [KA15], seawater desalination with hydrogels [HRK+13], the simulation of membrane proteins [RIH+07], and polymer translocation [SS17]. ESPRESSO is parallelized using the *message passing interface* (MPI) and can be used on several thousand cores. Some features, as e.g. the lattice-Boltzmann solver, can exploit the benefits of a single graphic accelerator card [RA12]. The software is intended to be highly flexible in order to cover a wide range of scenarios and to appeal to both beginners and experts. To achieve this, the ESPRESSO package is split into two parts: the core module and an interpreter. The core module consists of efficient implementations of algorithms needed for the particle and fluid simulation. The software is controlled by a *Tcl* or *Phython* interpreter interface that allows a fine tuning of the simulation and its parameters.

2.2 Tree-Based Grids Along Space-Filling Curves Using P4EST

The main concepts in the context of spatial adaptivity are unstructured and tree-based grids. Although unstructured grids are more flexible and a generalization of most other structures, they are very memory consuming and algorithmically complex to process. A commonly used data structure for three-dimensional information is the so called *octree*. Octrees are a natural generalization of regular grids and their hierarchical structure allows an adaptive refinement in areas of interest. As the name indicates, octrees either consist of inner nodes with exactly eight children or leaf nodes with no descendants. Typically, the children of a node subdivide the three-dimensional volume occupied by the parent into eight equally sized cubes. As for the mere partitioning of a given cube, it is sufficient to store only the leaf nodes of the tree, as inner nodes fully overlap with their decedents. A single regular tree equally divides each dimension by a power of two, depending on its level. The two-dimensional equivalent of octrees are *quadtrees*. The cells of grids based on octrees or quadtrees are called *octants* or *quadrants*, respectively. However, in this work both names are used interchangeably. Combining several octrees by linking their faces, edges, and corners allows the representation of more complex geometries. Those interconnected tree constructs are referred to as *forest*. The P4EST software library [BWG11] provides methods for managing such tree-based structures on highly parallel and distributed systems. The linearized representation of the tree leafs in a depth-first order is directly associated with space-filling curves, due to the self-similarity of both the tree-structure and the space-filling curve. Figure 2.1 shows a forest of spatially refined grids in two dimensions and their corresponding trees together with the z-order Morton curve [Mor66].

It has been shown that P4EST scales to over 450.000 cores [IBWG15] and it is even used for applications on over 3 million processes [MKM+15]. The parallel structure is divided among the processes such that each process only stores a consecutive part of all cells with respect to the space-filling order. Figure 2.2 visualizes this domain decomposition in the evenly distributed case on three processes. As can be seen, subdomains of the space-filling domain decomposition have no regular structure. The topology of all trees and their interconnection is stored in the `p4est_connectivity` data structure. In this work, however, only simulations within cuboidal domains are considered. Thus, it is sufficient to use `p4est_brick` which creates a regular structured connectivity with as well as without a periodic linking. The `p4est_mesh` and `p4est_ghost` data structures provide random access to any local cell's neighborhood no matter if those cells are local or ghost cells, have a different mesh width, or reside in a adjacent tree with a potentially different local coordinate system. The neighborhood enumeration is given by the scheme shown in Figure 2.3. Besides managing the structural information of the grid, the P4EST software library is able to handle and communicate data logically assigned to the cell as well. Thus, it is a powerful tool to extend existing scientific computation software based an regular Cartesian grids by the spatial adaptivity of tree-based grids.

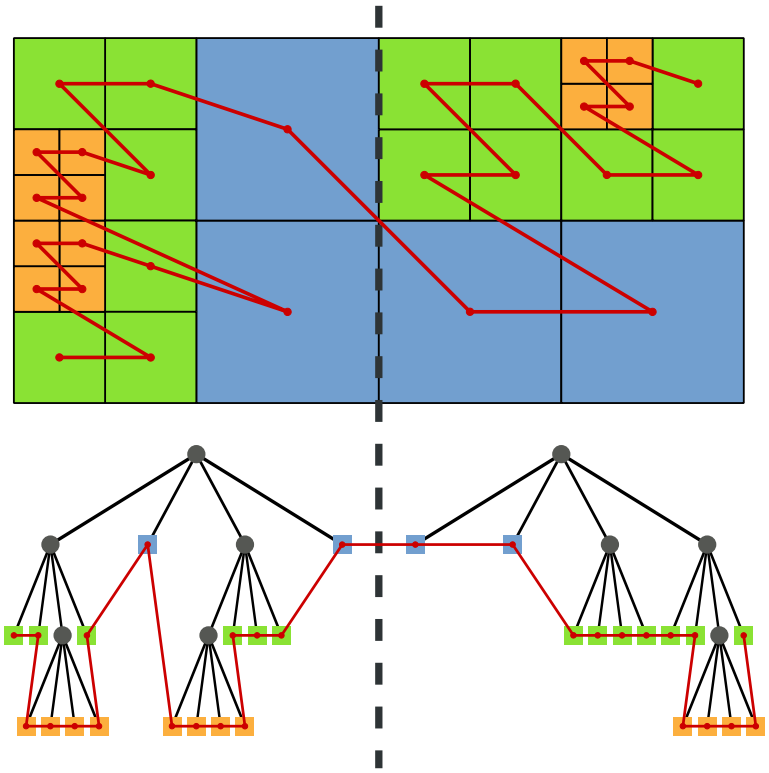


Figure 2.1: A forest of two spatially refined quadtrees (bottom) and the corresponding grids (top). Leaf nodes of the tree are colored with respect to their level. The Morton curve (red line) is given by a depth first order of the leaf nodes. Inner nodes (grey circles) do not appear in the linearized Morton curve.

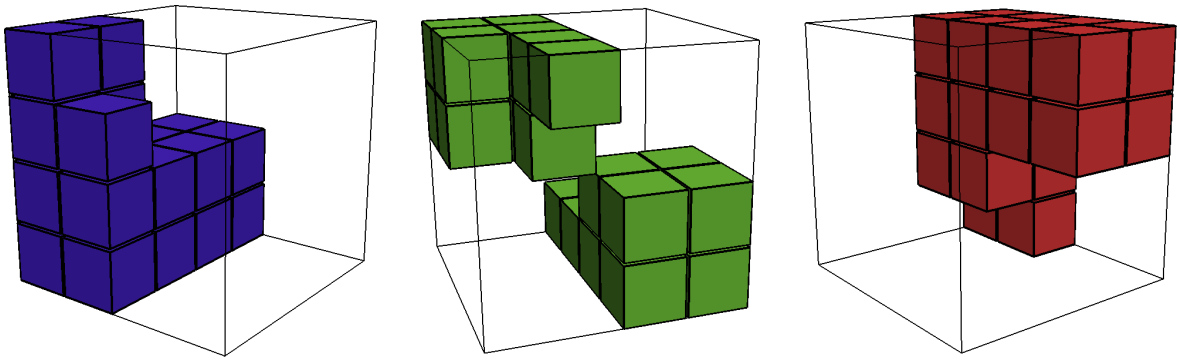


Figure 2.2: Domain decomposition on three processes (blue, green, red) of a regular octree on level two using P4EST. Due to the space-filling Morton order, subdomains have no regular shape. Even subdomains consisting of unconnected parts are possible, as seen in the center for the green process.

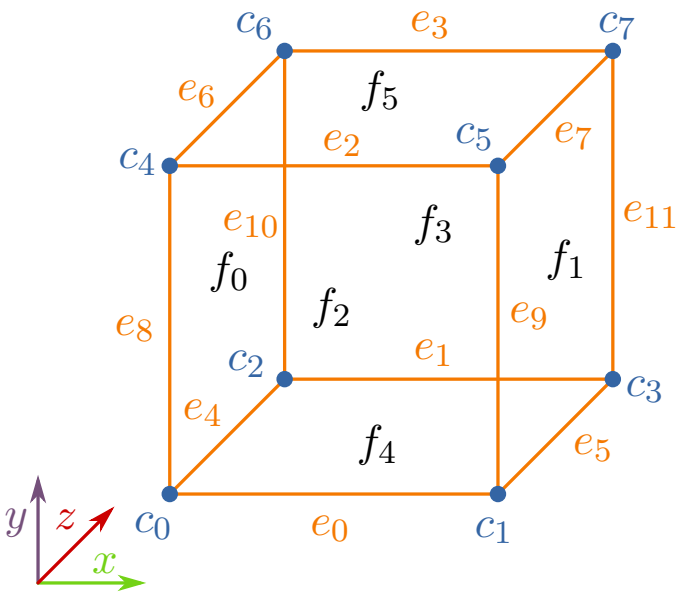


Figure 2.3: Enumeration scheme of structural information in P4EST. Face, edges, and corners are shown in black, orange, and blue, respectively. The linearized zero-based numeration starts with all faces continued by the edges and corners, thus it is $f_0 = 0$ and $c_7 = 25$.

The linearized Morton order for the grid cells allows efficient algorithms based on the space-filling curve. For any multi-dimensional integer value the z-curve index is computed by interleaving the bits for each dimension as shown in Figure 2.4 and Algorithm 2.1. Note that

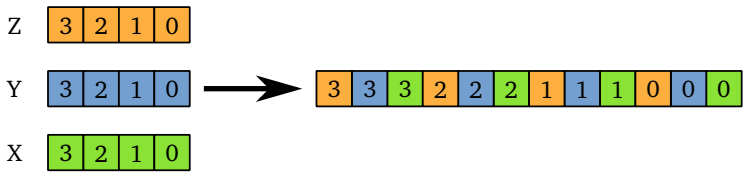


Figure 2.4: Bit interleaving for the space-filling Morton index. The three values (*X* green, *Y* blue, and *Z* orange) are combined to the Morton index on the right starting with the least significant bit.

some processor architectures provide bit manipulation intrinsics (e.g. the *parallel bit deposit (PDEP)* of the *BMI2* instruction set) that can be used for a simplified and efficient computation of the Morton index. For our purposes it is sufficient to use the Morton index for integers, since floating point values only need to be mapped to cells without distinguishing them further.

Algorithm 2.1 Computes the Morton index of a multi-dimensional integer value by interleaving the bits of each dimension.

```
1:  function MortonIndex(int[ ] x)
2:      idx ← 0
3:      pos ← 1
4:      while x[•] ≠ 0 do                          ▷ Repeat until no bits are left
5:          for d ∈ range(dim) do
6:              if mod(x[d], 2) = 1 then           ▷ Interleave lowest bit in this direction
7:                  idx ← idx + pos
8:              end if
9:              x[d] ← ⌊x[d] / 2⌋                  ▷ Shift values
10:             pos ← 2 · pos
11:         end for
12:     end while
13:     return idx
14: end function
```

Thus, floating point positions are converted to integers by dividing with the respective mesh width of the grid. Afterwards, the rounded quotients can be linearized using Algorithm 2.1.

One missing function of the P4EST library for the later discussed particle simulation in Chapter 4 and the particle-fluid coupling in Chapter 5 is the mapping of three-dimensional particle positions to the linear index of the containing grid cells. Although the cells are ordered along the Morton curve, the global cell enumeration is not equal to the Morton index of their three-dimensional Cartesian grid coordinate, due to adaptivity, non-cubic grid topologies, or grids that are not based on powers of two. A simple example for this problem is shown in Figure 2.5. In other words, the cell index used by the P4EST library is not directly coupled to the three-dimensional Cartesian coordinate of the grid. Algorithm 2.2 computes the global Morton index for a grid quadrant based on a reference level and the spatial position given at the lower front-left corner of the cell. This algorithm is later used as mapping to compare the area occupied by grid cells with particle positions.

Algorithm 2.2 Computes the global Morton index for a P4EST quadrant and a given reference level. Note that this index is not necessarily equal to the global P4EST index.

```
1: function GlobalQuadIdx(quad q, int ref_level)
2:     coord ← p4est_quad_vertex(q)                ▷ Get global coordinate for quadrant
3:     pos ← ⌊coord · 2^ref_level⌋                 ▷ Map Cartesian position to cell coordinate
4:     return MortonIndex(pos)
5: end function
```

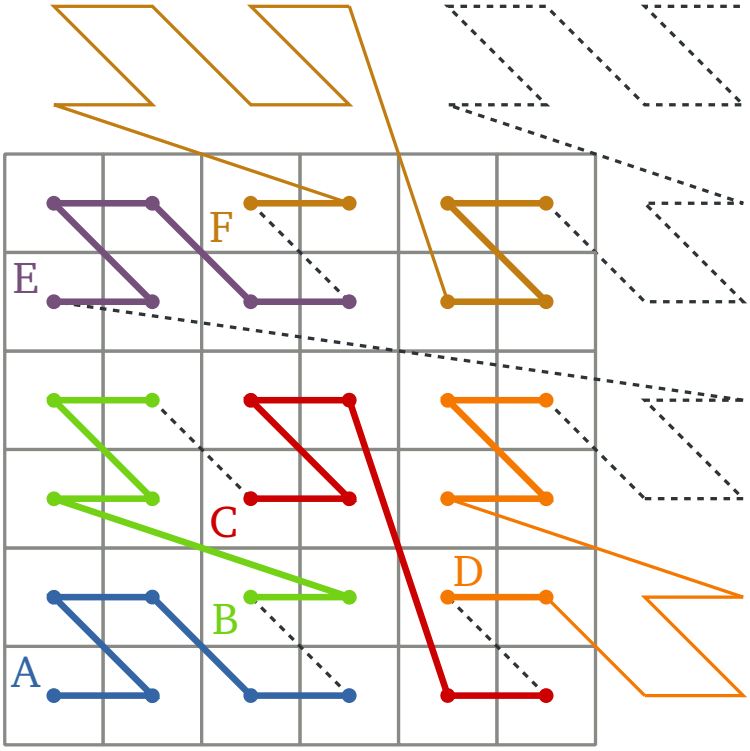


Figure 2.5: Regular grid with six cells in both directions and the corresponding global Morton curve. The grid is divided among six processes (*A, B, C, D, E, F*) such that each process has six local cells. The first cell of each process is marked by its letter. Process *A, B, C*, and *E* are continuous with respect to the global Morton index. The P4EST cell enumeration on process *D* and *F* has a gap with respect to the global Morton index, due to the grid's topology. The local cell index for process *A, B, C*, and *E* can be computed from the global Morton curve index. For process *D* and *F* a binary search has to be used to access a cell by its global Morton index.

CHAPTER 3

Fluid Simulation

One is often interested in the macroscopic behavior of fluids or gases rather than in intermolecular interactions. However, the representation of fluids based on molecules has a severe influence on the total simulation time as well as on the observable physical volume, because of an enormous computational workload that comes along with growing simulations. The goal of fluid dynamics is a change of scale from the microscopic particle view to a macroscopic continuum perspective of the fluid. Pioneers in the field of computational hydrodynamics were Navier, Stokes, and Boltzmann in the 19th century. Their work resulted in the famous and still used *Navier-Stokes* and *Boltzmann* equations for fluid kinetics. The Navier-Stokes equations are a coupled non-linear differential system for macroscopic fluid quantities as e.g. density, flow velocity, or pressure. The Boltzmann equation is a more general model for fluids and in the limit of low Mach numbers the Boltzmann equation is equivalent to the full Navier-Stokes equation [DL09, Deg04]. This section covers a short review on the *lattice-Boltzmann* (LB) method numerically solving the Boltzmann equations that arise from a probability density approach in the seven dimensional phase-time space. As the lattice-Boltzmann method is of such importance in scientific computations and still is a matter of research, the equations and its derivation are well covered in many works, as for example in [Sch08, Raa04, RDL77, Röh11] to name just a few. The lattice-Boltzmann method is usually motivated from the perspective of hydrodynamics with the *lattice gas automata* (LGA). However, in this chapter the sketch of a more analytical approach to derive the Boltzmann equations is shown as explained in more detail in [BGL91, AD69, RG65]. A formal analytical derivation of the Boltzmann equation is given in the review [Deg04].

The motion of each particle in a multi-body system with the particle mass m obeys the classical kinetic laws of ordinary differential equations

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}(t) &= \frac{1}{m}\mathbf{F}(\mathbf{r}(t)) \\ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}(t) &= \mathbf{v}(t) \end{aligned} \tag{3.1}$$

proposed by Newton. The time dependent particle position $\mathbf{r}$ and velocity $\mathbf{v}$ are evolved under the influence of the for this purpose not further specified force field $\mathbf{F}$. Applying Liouville's

theorem to the six dimensional flow map for the system (3.1) in the phase-space for position and velocity leads to the basic partial differential equation of motion

$$\partial_t f(t,\mathbf{r},\mathbf{v}) + \mathbf{v}\cdot\nabla_{\mathbf{r}} f(t,\mathbf{r},\mathbf{v}) + \frac{1}{m}\mathbf{F}(\mathbf{r})\cdot\nabla_{\mathbf{v}} f(t,\mathbf{r},\mathbf{v}) = 0 \tag{3.2}$$

with a continuum assumption for the substance. The Liouville equation (3.2) prescribes the time dependent evolution of the probability density function f in the force field $\mathbf{F}$ without any special inter-fluid interactions. The collision of particles or the interaction with the background medium can be modeled by introducing additional terms to the balance equation. This yields the extended, more general form

$$\partial_t f(t,\mathbf{r},\mathbf{v}) + \mathbf{v}\cdot\nabla_{\mathbf{r}} f(t,\mathbf{r},\mathbf{v}) + \frac{1}{m}\mathbf{F}(\mathbf{r})\cdot\nabla_{\mathbf{v}} f(t,\mathbf{r},\mathbf{v}) = \mathcal{G}[f] - \mathcal{L}[f] = \mathcal{Q}[f] \tag{3.3}$$

where $\mathcal{G}$ and $\mathcal{L}$ are gain and loss operators, respectively. In case of the Boltzmann equation it is, however, convenient to combine these additional terms to a single collision operator $\mathcal{Q}$.

The hard sphere model is used in order to formulate the collision term for the Boltzmann equation. In the hard sphere model the elastic collision of spheres is based on the conservation of momentum and energy. For two interacting particles, this gives the post-collision velocities $\mathbf{v}'$ and $\mathbf{v}_1'$ as

$$\begin{aligned} \mathbf{v}' &= \mathbf{v} - \langle \mathbf{v}-\mathbf{v}_1, \mathbf{n}\rangle \mathbf{n} \\ \mathbf{v}_1' &= \mathbf{v}_1 + \langle \mathbf{v}-\mathbf{v}_1, \mathbf{n}\rangle \mathbf{n} \end{aligned} \tag{3.4}$$

for the pre-collision velocities $\mathbf{v}$ and $\mathbf{v}_1$, respectively. The vector $\mathbf{n}$ denotes the collision normal. The particle motion based on the hard sphere collision is reversible in time, due to the reversibility of the system (3.1). To apply this discrete scheme to the probability density function in the general Liouville equation (3.3) the collision operator (3.4) has to be considered in the limit for small sphere diameters and large numbers of particles such that the mean free path of particles between collisions remains constant as explained in [Cer88, Deg04]. This boundary value analysis results in the collision operator

$$\mathcal{Q}[f] = \kappa \int\limits_{\mathbf{v}_1\in\mathbb{R}^3} \int\limits_{\mathbf{n}\in S_+^2} \Big|\langle \mathbf{v}-\mathbf{v}_1, \mathbf{n}\rangle\Big| \Big(f(t,\mathbf{r},\mathbf{v}')f(t,\mathbf{r},\mathbf{v}_1') - f(t,\mathbf{r},\mathbf{v})f(t,\mathbf{r},\mathbf{v}_1)\Big)\, \mathrm{d}\mathbf{n}\, \mathrm{d}\mathbf{v}_1 \tag{3.5}$$

for the Boltzmann equation with the density coefficient κ. The collision cross section S_+^2 is given by

$$S_+^2 := \Big\{ \mathbf{n}\in\mathbb{R}^3 \;:\; |\mathbf{n}| = 1\,,\; \langle \mathbf{v}-\mathbf{v}_1, \mathbf{n}\rangle > 0 \Big\} \tag{3.6}$$

and the post-collision velocities in (3.5) are computed with the relation (3.4). The Boltzmann equation (3.3) with the collision operator (3.5) is not reversible in time anymore, due to the limit consideration.

The probability density function f obtained by solving the Boltzmann equation is related to the characteristic macroscopic fluid quantities: mass density, velocity density, and energy density. They are given by

$$
\begin{aligned}
\rho(t,\mathbf{r}) &= m \int\limits_{\mathbf{v}\in\mathbb{R}^3} f(t,\mathbf{r},\mathbf{v})\,\mathrm{d}\mathbf{v} \\
\rho\mathbf{u}(t,\mathbf{r}) &= m \int\limits_{\mathbf{v}\in\mathbb{R}^3} \mathbf{v} f(t,\mathbf{r},\mathbf{v})\,\mathrm{d}\mathbf{v} \qquad (3.7)\\
\rho e(t,\mathbf{r}) &= m \int\limits_{\mathbf{v}\in\mathbb{R}^3} \frac{|\mathbf{v}|^2}{2} f(t,\mathbf{r},\mathbf{v})\,\mathrm{d}\mathbf{v}
\end{aligned}
$$

as the first three statistic moments of the distribution function with respect to the velocity. These are the only three invariants of the collision operator (3.5) and they correlate with the conservation laws for mass, momentum, and energy. The collision invariants motivate to consider the equilibrium state of the fluid. The equilibrium is a local state of the Boltzmann equation (3.3) with the operator (3.5) which is independent of time and space. It is given by the Maxwell distribution function

$$
f^{eq}_{\rho,\mathbf{u},T}(\mathbf{v}) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \frac{\rho}{m} \exp\left(-\frac{m(\mathbf{v}-\mathbf{u})^2}{2k_B T}\right) \qquad (3.8)
$$

for the macroscopic values $\rho, \mathbf{u}$, and T as density, mean velocity, and temperature, respectively. The Maxwell equilibrium distribution fulfills the relation $\mathcal{Q}[f^{eq}] = 0$ for the collision operator. With the Maxwell distribution any arbitrary probability density function can be split into

$$
f = f^{eq} + f^{neq} \qquad (3.9)
$$

where f^{neq} is the difference to the equilibrium state. For distributions that are close to the equilibrium the collision operator $\mathcal{Q}$ can be linearized, in order to apply the splitting (3.9). The linear BGK operator

$$
\mathcal{Q}[f] = \frac{1}{\nu}(f^{eq} - f^{neq}) = -\frac{1}{\nu} f^{neq} \qquad (3.10)
$$

is named after Bhatnagar, Gross, and Krook [BGK54] and often used for the Boltzmann equation and in the discretized lattice-Boltzmann method. The quantity ν^{-1} is related to the mean frequency for an arbitrary distribution to change into the equilibrium state.

3.1 Lattice-Boltzmann Method

Altogether the Boltzmann equation (3.3) for a force free setting, i.e. $\mathbf{F} \equiv 0$, with the BGK operator (3.10) is given by

$$\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{r}} f = -\frac{1}{\nu}(f - f^{eq}) \ , \text{ in } (0,\infty) \times \mathbb{R}^3 \times \mathbb{R}^3 \ . \tag{3.11}$$

Directly solving this equation is quite cumbersome due to its high dimensionality. Thus, a numerical scheme is required to tackle this task. This comes along with a discretization of the seven dimensional phase-time space. In a first step the velocity space is discretized with the help of the Hermite polynomial representation of the probability density function and the Gauss quadrature [Sch08, SH98]. The set of Boltzmann equations for discrete velocities $\mathbf{c}_i$ reads as

$$\partial_t f_i + \mathbf{c}_i \cdot \nabla_{\mathbf{r}} f_i = -\frac{1}{\nu}(f_i - f_i^{eq}) \ , \text{ in } (0,\infty) \times \mathbb{R}^3 \tag{3.12}$$

where f_i are functions only in space and time. The solutions f_i represent virtual particle populations with respect to the velocities $\mathbf{c}_i$. The Maxwell distribution of the equilibrium state is transferred in the same way using the Hermite polynomial expansion. To obtain the equilibrium distribution for low Mach numbers and with respect to the discrete velocities, it is written as second order expansion

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

with the coefficient ω_i and the speed of sound c_s. The hydrodynamic quantities can still be written quite similar to (3.7) by

$$\begin{aligned} \rho &= \sum_i f_i \ , \\ \rho \mathbf{u} &= \sum_i f_i \, \mathbf{c}_i \ , \\ \Pi &= \sum_i f_i \, \mathbf{c}_i \otimes \mathbf{c}_i \ , \end{aligned} \tag{3.14}$$

as sum over the mesoscopic particle populations.

Equation (3.12) with the discrete velocities $\mathbf{c}_i$ is rewritten as ordinary differential equation in order to fully discretize it in space and time. The first order approximate integral of this ODE yields the lattice-Boltzmann equation

$$f_i(t + \tau, \mathbf{r} + \tau \mathbf{c}_i) = f_i(t, \mathbf{r}) + \mathcal{L}_{ij} \Big[f_j(t, \mathbf{r}) - f_j^{eq}(t, \mathbf{r}) \Big] \tag{3.15}$$

with the general linear operator $\mathcal{L}_{ij}$ which is in case of the BGK collision equal to $-\nu^{-1}\delta_{i,j}$. As seen, the discretized phase space is coupled by the time step τ. Typically, the velocities $\mathbf{c}_i$ are chosen such that particles are only transported to directly adjacent lattice cells. For a three-dimensional cubic lattice the *D3Q19* model seen in Figure 3.1 is widely used. The scheme consists of 19 discrete velocities connecting the cell with itself, its six face and twelve edge neighbors.

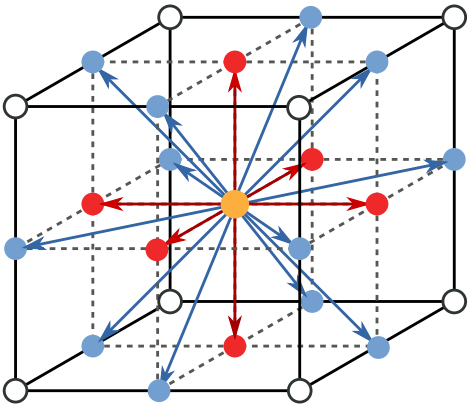


Figure 3.1: The D3Q19 scheme for the lattice-Boltzmann method [Sch08]. There are 19 discrete velocities: one to the cell itself (orange), six to the face neighbors (red), and twelve to the neighbors connected over edges (blue).

Algorithmically, the lattice-Boltzmann method can be split into two main parts: a) a cell-local collision step in which the given set of pre-collision particle populations of all discrete velocities is relaxed towards their local equilibrium; b) a streaming step, where the relaxed post-collision populations are transported to the adjacent neighboring cells with respect to the velocity $\mathbf{c}_i$ to become the new pre-collision populations for the next integration step. The algorithmic steps are sketched schematically in Figure 3.2.

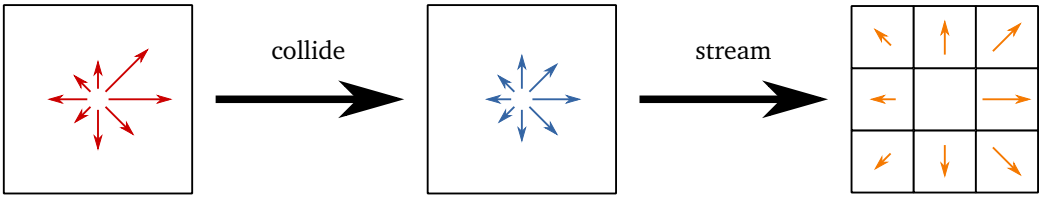


Figure 3.2: Regular 2D grid showing the schematic collision and streaming step of the lattice-Boltzmann method. The discrete velocities are indicated by the arrows, where the length illustrates the corresponding particle population. Pre-collision populations (red) are relaxed towards their local equilibrium during the collision step. In the streaming step the relaxed populations (blue) are propagated to the respective adjacent cells as new pre-collision values (orange) for that direction.

3.2 Adaptive Refinement Using Octrees

A big issue with the classical lattice-Boltzmann method is the regular cubic grid upon which it operates. The existence of e.g. obstacles, boundaries, or phase surfaces within the simulated volume usually requires a very fine resolution for the discretization. Simply reducing the mesh width of the grid, however, not only has a severe influence on the total number of lattice cells as it scales with $\mathcal{O}(h^{-3})$, but also forces the time integration to perform finer steps due to the coupled discretization of the phase space. Thus, the runtime of a simulation for a fixed time interval increases drastically by regularly refining the grid. A common approach to this problem is the local adaptive refinement of the grid in areas of interest. One of the main attributes of spatially adaptive grids is the existence of neighboring cells with different mesh sizes. Nevertheless, the lattice-Boltzmann method strongly depends on the regular structure of the grid, because of the previously discussed discretization scheme, as e.g. the D3Q19 stencil. In 2006 Rohde, Kandhai, Derksen, and van den Akker proposed a method for lattice-Boltzmann schemes on locally refined grids [RKDA06]. Together with a spatial representation of the grid based on octrees using the P4EST library, Lahnert et al. developed an adaptive lattice-Boltzmann method that is minimally invasive integrated in the ESPRESSO simulation software [LBH$^+$16].

The P4EST based cell structure for the lattice-Boltzmann discretization uses a 2:1 balancing. I.e. neighboring cells are only allowed to have a level difference of at most one. This simplifies the adaptive lattice-Boltzmann method as a cell can either interact with same sized cells or cells with twice or half the mesh width. For simplicity reasons it is assumed that the grid only has two different levels for now. This is sufficient to fully describe the method, since interactions with both smaller as well as bigger cells occur and the method can be applied recursevely for grids with more than two levels. The proposed method for adaptive grids uses the same microscopic velocity discretization for all cells regardless of their size. The integration time step for coarse cells thus is twice the time step of fine cells. In other words, the fine part of the gird has to perform two full collide-stream cycles before one coarse time step can be computed. Extending this scheme for grids with multiple refinement levels leads to a W-shaped subcycling scheme as illustrated in Figure 3.3. Depending on the subcycle state, the collision and streaming operations are performed on different levels. The method presented in [LBH$^+$16] uses the finest level as reference for the lattice-Boltzmann scheme. Thus, a single integration step of the simulation equals one step of the subcycle. The resulting integration algorithm invokes the collision and streaming level based. The collision is performed in increasing level order, thus from coarse to fine with respect to the current subcycle step while the streaming operates vice verse in descending level order.

In terms of spatial adaptivity the collision step is unproblematic, because it is a purely cell-local operation. For each cell the pre-collision particle populations are transferred into macroscopic fluid modes and relaxed towards the local hydrodynamic equilibrium. The collision operation, however, interacts with neighboring cells. Hence, cells at the refinement boarder have to

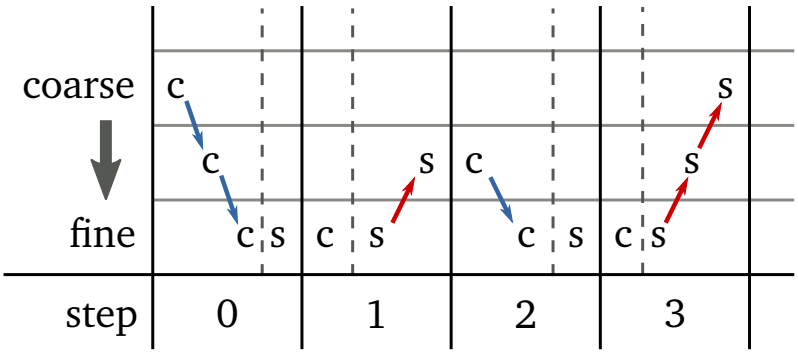


Figure 3.3: Subcycling scheme for the adaptive lattice-Boltzmann method and a grid on three levels (coarse to fine from top to bottom). The letters represent the collision (*c*) and streaming (*s*) operation on the respective level. Both operations are divided by a dashed line for each step of the subcycling scheme. Blue arrows indicate the prolongation from coarse to fine virtual cells. Red arrows show the restriction operation from virtual cells to their parent.

propagate their results to cells with twice or half the mesh width. This is achieved by embedding virtual child quadrants into coarse cells at refinement boundaries. After the collision operation on such a coarse level cell, the computed post-collision populations are divided among the virtual sub quadrants by a prolongation operator. The streaming step on the fine level propagates the information from virtual cells to real quadrants, thus from a coarse to a fine cell. Post-collision values are transported from real fine cells to virtual quadrants in the same way. Before the streaming on the coarse level is invoked in the last step of the subcycling scheme, all populations from virtual quadrants are combined and passed to the parent by a restriction operator. This streams the post-collision information from fine to coarse cells. Figure 3.4 schematically shows the subcycle scheme with the restriction and prolongation operation on virtual cells. Thus, streaming is performed on both real and virtual cell for each level whereas the collision operation is only used for real cells.

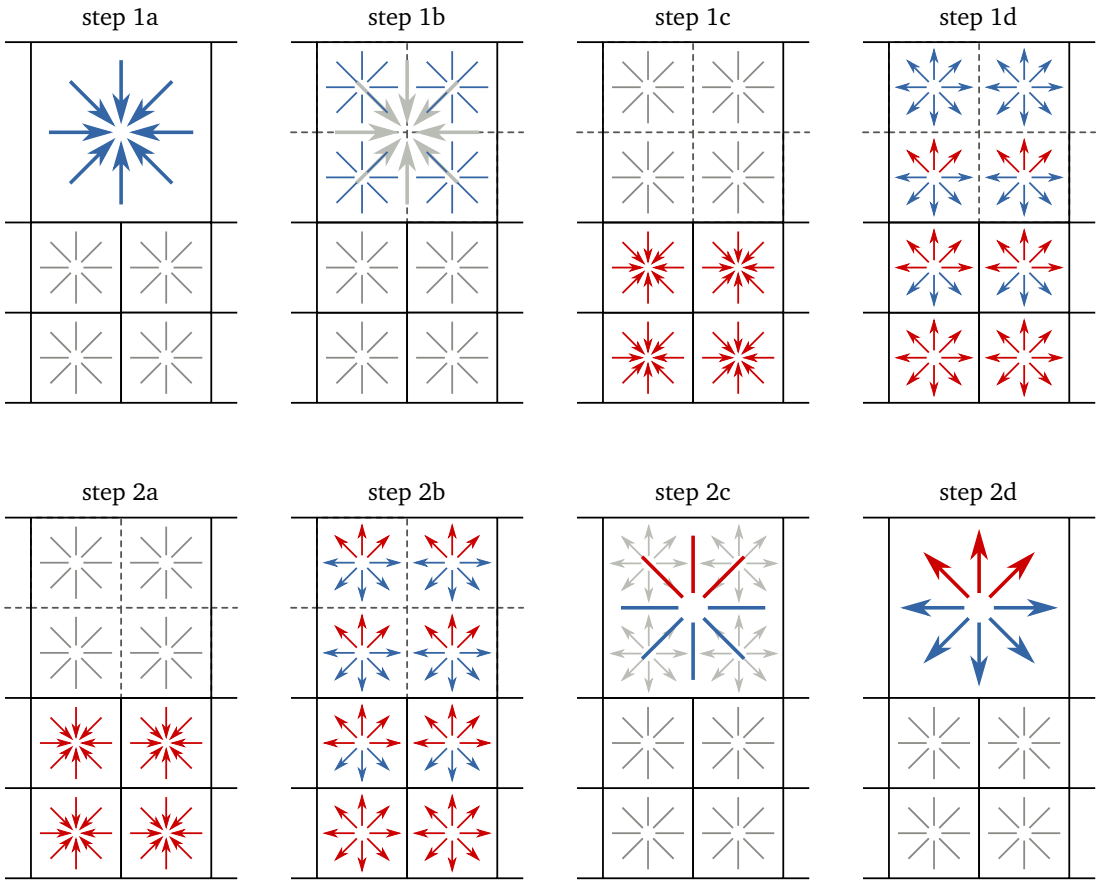


Figure 3.4: Steps of the subcycling for virtual cells. Red and blue arrows correspond to values that are originated from fine and coarse cells, respectively. Arrows pointing inwards represent population directly after a collision. Arrows pointing outwards illustrate populations after the streaming operation. Gray populations are not changed in the corresponding substep. The subcycling for two different levels has two steps, each divided into four parts: 1a) collision on the coarse grid; 1b) prolongation to the virtual cells; 1c & 2a) collision on the fine grid; 1d & 2b) streaming on the fine grid; 2c) restriction from virtual cells to the parent; 2d) streaming on the coarse grid.

CHAPTER 4

Particle Simulation

The molecular dynamics (MD) particle simulation is a powerful and important tool in scientific computing. The method was developed in the late 1950s [FPU55, AW59] as field of theoretical physics. Over the past decades the significance of MD simulations steadily increased hand in hand with the rapid development of computer technology. The simulation analyzes the physical movement of atoms, molecules, or arbitrary particles by numerically solving Newton's equations of motion. The particles are coupled by molecular interaction potentials and other external force fields. Thus, it results in a classical multi-body system. The concept of molecular dynamic simulations is straight forward. The trajectories of all particles are evolved individually with Newton's second law of motion by computing the interactive forces. But as simple as the idea of molecular dynamics is in theory, as challenging the method is in practice, due to the sheer amount of particles and interactions. Molecular dynamics nowadays have a wide range of applications, e.g. computational chemistry, material science, or biomolecular engineering. As the full trajectory information of all particles is available the method is used for structure analysis of dissolved polymers as proteins [SO99, KK05, SAK02]. The change and behavior of macroscopic quantities such as the free Helmholtz energy, temperature, or pressure [JR85, HEHB15, FS02] is also of interest. These observables are build by statistical means over all particles and the simulation time as in the *free energy perturbation* method [CP07]. There are a lot of software packages and libraries dedicated to highly parallel MD simulations as they are such a wide and important field in science. E.g. *ls1 mardyn* [NBB+14], *NAMD* [PBW+05], *GROMACS* [HKVDSL08], *Desmond* [BCX+06], *CHARMM* [BCX+06], *Amber* [SFCW13], *LAMMPS* [Pli95], or the here used ESPRESSO [LAMH06, ALK+ed] code.

The coupled system of ordinary differential equations directly resulting from Newton's laws of motion for the N-body problem is given by

$$
\begin{aligned}
\mathbf{F}_{pot,i}(\mathbf{r}_1, ..., \mathbf{r}_N) &= -\nabla_{\mathbf{r}_i} U(\mathbf{r}_1, ..., \mathbf{r}_N) \\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_i(t) &= \frac{1}{m_i}\Big(\mathbf{F}_{pot,i}(\mathbf{r}_1(t), ..., \mathbf{r}_N(t)) + \mathbf{F}_i\Big) \quad , \ t \in (0, \infty), \ \forall\, i \\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_i(t) &= \mathbf{v}_i(t) \quad , \ t \in (0, \infty), \ \forall\, i \\
(\mathbf{r}_i(0), \mathbf{v}_i(0)) &= (\mathbf{r}_{i,0}, \mathbf{v}_{i,0}) \quad , \ t = 0, \ \forall\, i
\end{aligned}
\tag{4.1}
$$

with $\mathbf{r}_i$ and $\mathbf{v}_i$ as position and velocity of the i-th particle, respectively. The particles are affected by the individual external force $\mathbf{F}_i$ and the interactive force $\mathbf{F}_{pot,i}$ based on the intermolecular potential U. In general, the potential is highly non-linear and there are many degrees of freedom as the number of particles needed for reasonable statistical means or representative polymer structures is very large. Adding additional stochastic forces, e.g. for Brownian motion, increases the complexity of the system (4.1) even more. Thus, numerical schemes are needed as the equation is quite tough or even impossible to solve analytically. System (4.1) can be solved iteratively by Runge-Kutta methods which are based on the series expansion of the differential equations using Taylor's theorem with respect to small time steps. A common method for classical mechanics and motion equations is the *leapfrog* integration [Ise86]

$$\begin{aligned} \mathbf{r}_i^n &:= \mathbf{r}_i^{n-1} + \mathbf{v}_i^{n-1/2}\Delta t \\ \mathbf{v}_i^{n+1/2} &:= \mathbf{v}_i^{n-1/2} + \frac{1}{m_i}\Big(\mathbf{F}_{pot}(\mathbf{r}_1^n, ..., \mathbf{r}_N^n) + \mathbf{F}_i\Big)\Delta t \end{aligned} \tag{4.2}$$

with the in time discretized particle quantities $\mathbf{r}_i^n$ and $\mathbf{v}_i^n$. It is a second-order method that requires the same number of force evaluations as the first-order Euler integration. The leapfrog integration conserves the energy of the system due to its symplectic properties. Furthermore, for constant Δt it is stable for oscillatory behavior characterized by the frequency ω as long as $\Delta t \leq 2/\omega$ is satisfied [BL04].

The potential U in Equation (4.1) resulting in the interactive force $\mathbf{F}_{pot,i}$ can be seen as sum over all possible k-body potentials U_k. This yields

$$\begin{aligned} U(\mathbf{r}_1, ..., \mathbf{r}_N) &= \sum_{k=1}^{N} \sum_{(\mathbf{x}_1,...,\mathbf{x}_k)\in\mathcal{P}_k(\{\mathbf{r}_i\}_{i=1}^N)} U_k(\mathbf{x}_1, ..., \mathbf{x}_k) \\ &= \sum_{i=1}^{N} U_1(\mathbf{r}_i) + \sum_{i=1}^{N}\sum_{j=i+1}^{N} U_2(\mathbf{r}_i, \mathbf{r}_j) + \cdots + U_N(\mathbf{r}_1, ..., \mathbf{r}_N) \end{aligned} \tag{4.3}$$

where $\mathcal{P}_k$ is the power set of all permutations with k elements. For a single potential U_k the number of possible combinations is given by the binomial coefficient. Hence, the total amount of all appearing k-body potentials in (4.3) is 2^N for the N-body problem. I.e. even small MD simulations with only 100 particles have a total of 2^{100} or about 10^{30} interactions. To grasp this huge number, lets assume that each interaction only needs one floating point operation (which is a certain lower bound), then a single integration step on the *Sunway TaihuLight* supercomputer (Rank 1 in the TOP500 in 2016 [Pro16]) with a peak performance of about $100\ PFlop/s$ would take more than $3 \cdot 10^5$ years. Therefore, most of the used and investigated potentials are two-body interactions. Neglecting all higher terms thus reduces the complexity of the potential stated in Equation (4.3) to $\mathcal{O}(N^2)$ which is still a huge issue in terms of algorithmic complexity. The remaining two-body potential can be written as

$$U_2(\mathbf{r}_i, \mathbf{r}_j) = U_2(\|\mathbf{r}_{i,j}\|) = U_2(r) \tag{4.4}$$

with $\mathbf{r}_{i,j}$ as distance vector between the i-th and j-th particle, due to symmetry reasons. In some cases it is useful to truncate the potential after a given cut-off radius as later shown in Section 4.1. The spatial restricted potential looks like

$$U_{r_c}(r) = \begin{cases} U_2(r) & , \ r \leq r_c \\ 0 & , \ \text{else} \end{cases} \tag{4.5}$$

where r_c is the cut-off radius and U_2 an arbitrary two-body potential. However, using U_{r_c} instead of U_2 leads the truncation error

$$e_{r_c} = \int_{\|\mathbf{x}\|>r_c} |U(\|\mathbf{x}\|)| \, \mathrm{d}\mathbf{x} = 4\pi \int_{r_c}^{\infty} |U(r)r^2| \, \mathrm{d}r \tag{4.6}$$

in the energy. By analyzing Equation (4.6) one can easily see that the truncation error e_{r_c} is bounded and

$$\lim_{r_c \to \infty} e_{r_c} = 0 \tag{4.7}$$

if and only if $U \in \mathcal{O}(r^{-p})$ with $p > 3$ holds. Using this observation, potentials can be divided into two groups: Potentials with a bounded error are called *short ranged* potentials as they can be truncated; Those with a diverging error term are called *long ranged* potentials. Long ranged interactions, as e.g. the Coulomb potential for two charged particles, need a different treatment and are further discussed e.g. in [GKZ07]. In general, truncating the potential as in Equation (4.5) produces a jump discontinuity at the cut. Shifting U_2 by a constant makes the truncated potential continuous. This has no effect on the interactive forces as the constant vanishes by differentiating the potential. Introducing a further term as described in [FS02] even allows the derivative of the potential, the interactive forces, to be continuous. This is important to minimize numerical errors at the cut-off radius.

One of the most famous intermolecular potentials that models the Van der Waals attraction as well as the Pauli repulsion is the short ranged *Lennard-Jones* or *12-6* potential given by

$$U_{LJ}(r) = 4\varepsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right) \tag{4.8}$$

as proposed by Lennard-Jones [Jon24] in 1924. It is often used in molecular dynamic simulations due to its computational simplicity. The Lennard-Jones potential can be fitted to experimental or theoretical data by the parameters ε and σ. The depth of the potential well is given by ε which influences the interactive forces in a linearly manner, while the parameter σ marks the zero-crossing of the function. Hence, σ correlates with the particle diameter in the hard sphere model. At $r_m = 2^{1/6}\sigma$ the function has its global minimum of $-\varepsilon$ and a derivative of zero. Thus, particles with a distance of r_m bond as they are neither attracted nor repelled. Figure 4.1 shows the normalized Lennard-Jones potential. It can be seen that for $r < r_m$ the slope is negative and thus particles that are closer than r_m repel each other. Particles with a distance greater than r_m are attracted due to the positive slope.

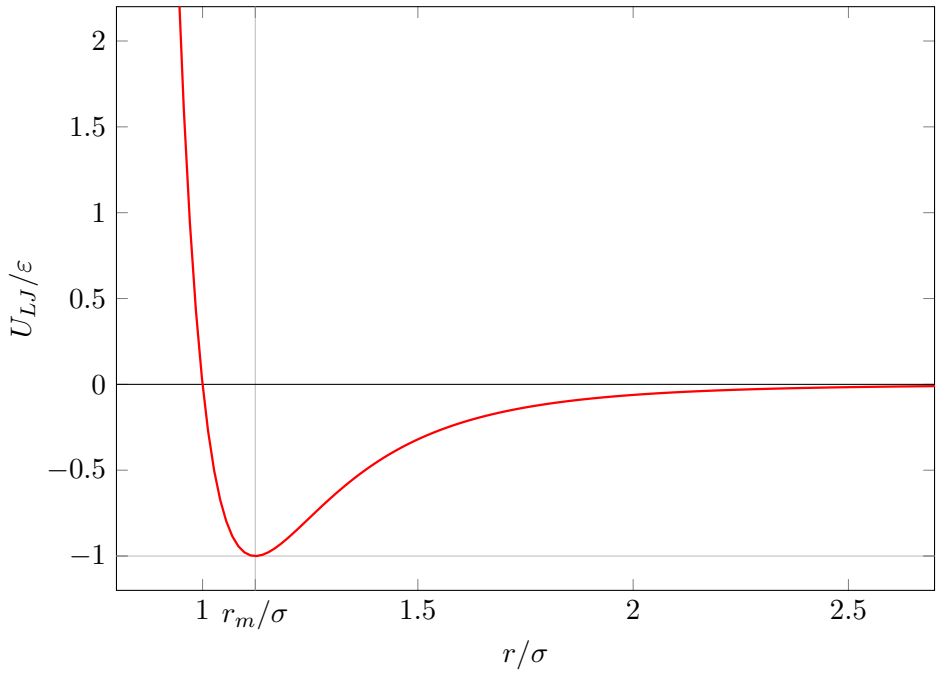


Figure 4.1: The normalized Lennard-Jones potential U_{LJ} as given in Equation (4.8). For $r = r_m := 2^{1/6}\sigma$ the potential has its global minimum of $-\varepsilon$.

4.1 Linked Cells Algorithm

Computing all two-body potentials within the system is in $\mathcal{O}(N^2)$ for N particles. Even for state-of-the-art computers this is a costly and unwanted factor in terms of runtime for very large simulations. A theoretical minimum for particle force computations is in $\mathcal{O}(N)$ as all particles need to be touched at least once to apply individual forces. Restricting the interacting potential of the multi-body system to the short-ranged case allows a truncation of the interaction radius with a bounded error. As explained in the following, this reduces the complexity to $\mathcal{O}(N)$ for homogeneously distributed particles.

In a homogeneous setting the amount of particles is bounded by a constant. Thus, each particle only has $\mathcal{O}(1)$ neighbors inside the given cut-off radius as long as the particle density is constant. This yields a complexity of $\mathcal{O}(N)$ for the computation of the interactive forces. However, the costs for finding all neighbors inside the cut-off radius for each particle with a naive approach still is in $\mathcal{O}(N^2)$ and even the efficient and parallel k-nearest neighbor method presented by Connor and Kumar [CK10] only achieves a complexity of $\mathcal{O}(N \log N)$ for this purpose. Thus, a commonly used method to compute short-ranged two-body potentials with minimal theoretical complexity is the so called *linked cells* or *cell lists* algorithm as described in [GKZ07, AT89, HGE74, Sch73] to name just a few. The basic principle of the linked cells algorithm is the partitioning of the domain into a regular grid with a mesh width in order of

the cut-off radius. Inserting an unordered set of particles into the grid is in $\mathcal{O}(N)$. With a mesh width greater or equal to the cut-off radius all potential interaction partners of a particle are either located in the containing cell or in the eight (2D) or 26 (3D) neighboring cells, the so called *full-shell*. This is illustrated in Figure 4.2 for a two-dimensional grid.

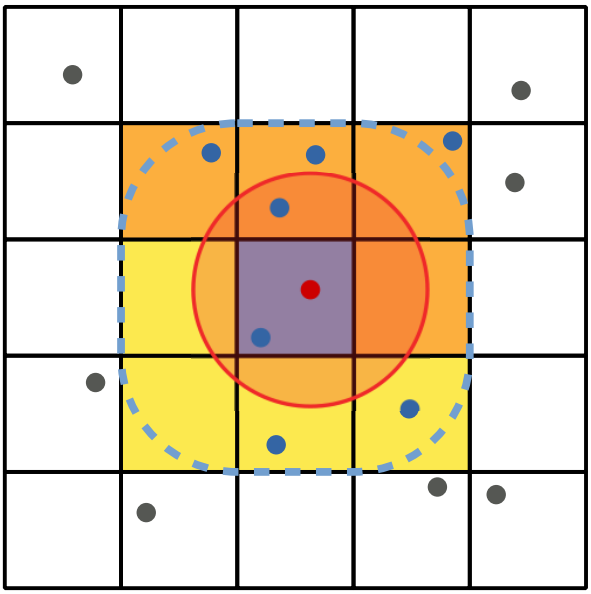


Figure 4.2: Linked cells grid with a mesh width equal to the cut-off radius. For the blueish center cell the full- and half-shell is highlighted. Both the yellow and orange cells are the full-shell where only one of both parts is the half-shell. The influential area of the center cell is marked with the blue dashed line. All blue particles are considered by the linked cells algorithm, but only those inside the red circle contribute to the interaction with the red particle.

Thus, it is sufficient to take only particles into account that are assigned to the same or adjacent cells, in order to compute the interactions. For a homogeneous distribution with a constant particle density this results in $\mathcal{O}(1)$ pairs for each particle, although even particles outside of the cut-off radius are considered this way. Newton's third law of motion states that each force has an equal counterforce in the opposite direction. Hence, a force resulting from a two-body potential on a particle can also be applied with inverted sign to the other respective particle. Algorithmically, this halves the amount of interactions that have to be computed, as a force is applied to both involved particles. To assure that all pair interactions are considered in total, the algorithm loops for each particle over all other particles in the same cell with a higher index and over all particles in the so called *half-shell*. The half-shell is half the number of all adjacent cells with the property that any two elements in the half-shell are not located point-symmetrically to the center. Both half- and full-shell are also shown in Figure 4.2 as highlighted cells. The amount of particles outside the cut-off radius can be reduced by choosing a smaller mesh width as seen in Figure 4.3. Although the cell's influential area is

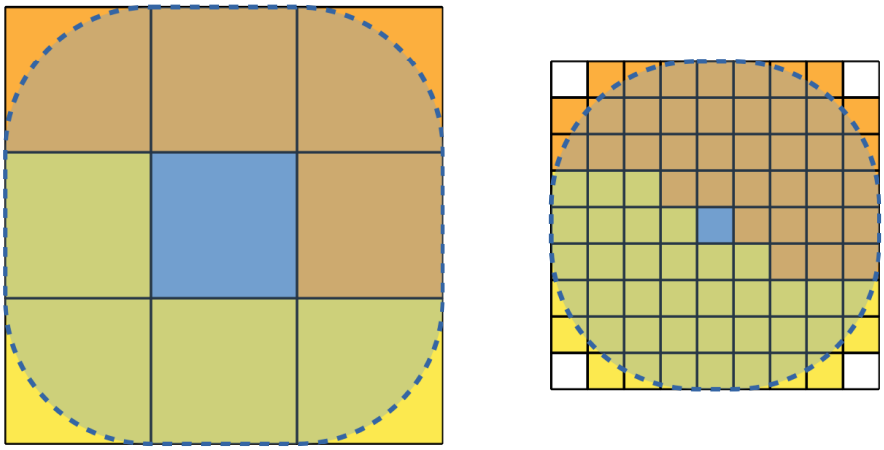


Figure 4.3: Left: Linked cells grid with the same mesh width as the cut-off radius. Right: Linked cells grid with 0.25 times the cut-off radius as mesh width. The influential area of the blue center cell surrounded by the blue dashed line covers more neighboring cells for finer meshes, but has less overhead. Full- and half-shell are marked in yellow and orange as in Figure 4.2.

better represented with finer cells, it covers more, not even directly adjacent, neighboring cells. Hence, the resulting half- and full-shell information is algorithmically more complex to determine and there is no fixed neighborhood stencil that can be used for simulations with arbitrary mesh widths. Buchholz [Buc10] even introduced a method for an adaptive mesh width based on the particle density. However, in the following we assume that the grid size is at least the cut-off radius, due to simplicity reasons.

The molecular dynamic simulation is bounded by a simulation box representing the observed physical volume. The choice of boundary conditions for this box depends on the simulated problem. Without any special conditions the particles behave as they are placed in a potential free vacuum. For a dispersing motion this limits the maximal simulation time, since the particles reach the boundary sooner or later. For a homogeneous setting periodic boundary conditions are practical where the simulated volume is patched as tile to virtually fill the whole space. It is not sufficient to simply wrap the neighborhood information around the boundary without destroying the method's property of implicitly satisfying the minimum-image convention as particles are always folded into the primary simulation box. Hence, the positions of periodic copies need to be shifted by the respective dimensions of the simulation box. The linked cells grid is extended by the so called *ghost layer* or *halo* around the domain. This layer has to be thick enough to completely cover the full-shell for all boundary cells. In our case it is a single layer as we restrict ourselves to the case of mesh widths greater than or equal to the cut-off radius. After each time integration step, the cells in the ghost layer are filled with shifted copies of particles in the respective boundary cell, the *mirror* cells. Figure

4.4 shows an exemplary mirror-ghost communication for periodic boundary conditions with both ghost and mirror cells highlighted. Additionally, the ghost layer simplifies the algorithms even in the non-periodic case where no particles are copied, since the same stencil can be used in the whole domain and all neighbors in the half-shell exist for boundary cells.

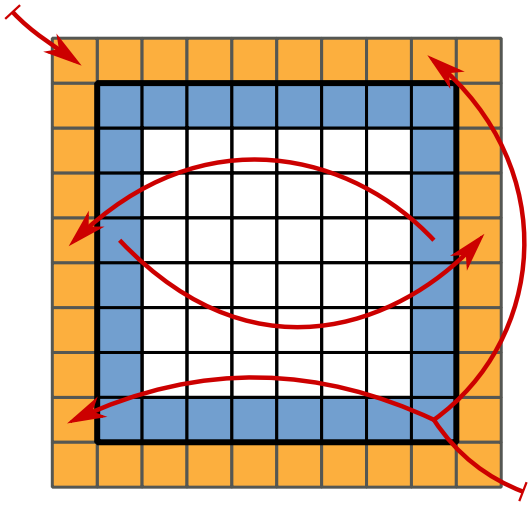


Figure 4.4: Exemplary communications of shifted particle data in a setting with periodic boundary conditions. The simulation box is surrounded by the bold black line. The shifted content of the blue mirror cells is copied to the orange ghost layer. Corner cells have more than one periodic image.

4.2 Regular Domain Decomposition

In molecular dynamics one is often interested in statistical means of certain particle properties [HEHB15, FS02], e.g. the average kinetic energy or the fluctuation in the temperature, rather than in the whole trajectories. Thus, MD simulations tend to be very large in order to have meaningful statistical quantities. I.e. a complete ab initio simulation of $1\ mol$ would need about $6.022 \cdot 10^{23}$ particles which exceeds the capability in terms of runtime and memory of all state-of-the-art super computers, c.f. [Pro16]. The molecular dynamic simulation has to be distributed and computed in parallel to make use of modern computer architectures and super computers. A commonly used method to parallelize the linked cells algorithm is the spatial domain decomposition, which divides the simulation box into subvolumes that are spread among the available processes. As almost every parallelization approach it comes along with additional communication, since the distributed sub-systems are coupled over the process boundaries. The regular domain decomposition uses cuboidal volumes which are aligned to each other to form a regular grid of subdomains. With such a decomposition it is straight forward to determine the process-particle membership. However, neighboring

processes have to exchange particle information for boundary cells. Similar to the ghost layer communication for periodic boundary conditions mentioned above, the content of mirror cells is sent and stored in the receiving process' ghost cells. For process-domain boundaries within the simulation box particles are not shifted. It is sufficient for each process to communicate only with neighboring processes connected over subdomain faces. Performing the communication in separate steps for each dimension propagates the information even over edges and corners as schematically seen in Figure 4.5.

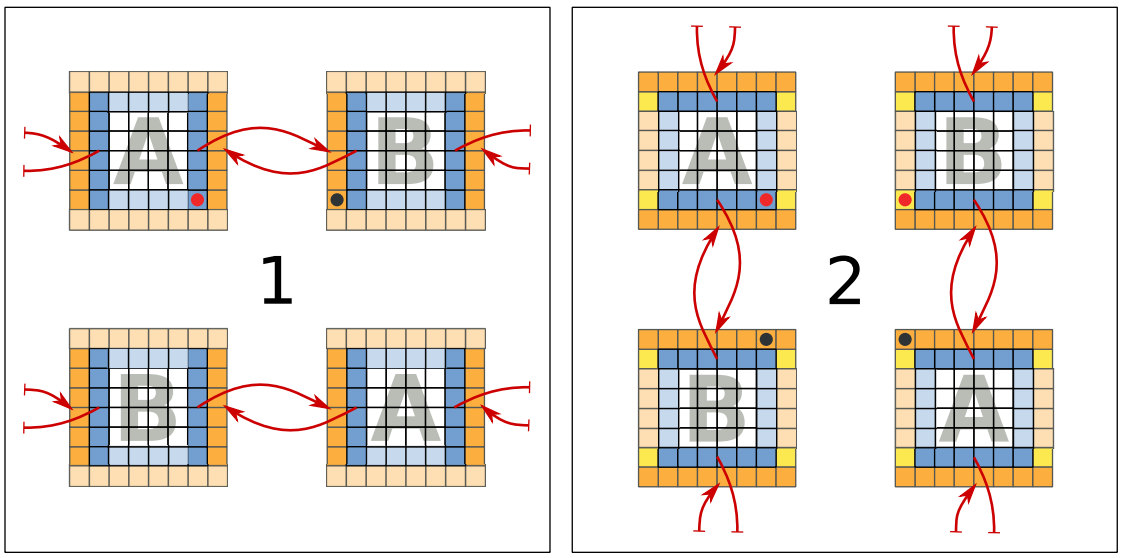


Figure 4.5: Schematic ghost layer communication for the parallel linked cells method. The first step on the left exchanges data along the first dimension. The exemplary particle of the top left subdomain is copied over the face to the right. In the second step on the right along the remaining dimension both the original and the copy are communicated to the ghost layers of lower subdomains. After both steps, the particle information is propagated over the domain corner from the top left to the bottom right process. Processes marked with *A* first perform the send and afterwards the receive operation. Processes marked with *B* operate vice versa.

The computation as well as the communication is invoked in parallel on all processes, but having two processes perform a send operation while waiting for the other process to receive yields in a communication deadlock. A checkerboard pattern is applied for the synchronous communication to prevent deadlocks and maximize the throughput. All processes assigned to one checkerboard color start the communication with the send operation while the other group begins with the receiving counterpart. Afterwards the scheme is applied vice versa. This is repeated for all dimensions. For a three-dimensional simulation this results in a communication routine with six substeps for each process which are performed after every time-integration step.

4.3 Domain Decomposition Based on Space-Filling Curves

The usual approach to parallelize the linked cells algorithm is a regular domain decomposition where the total simulation volume is divided into several cuboids that are distributed among the available processes. The regular structure allows an easy communication and mapping scheme for the particles. However, the shape and size of subdomains strongly depends on the number of used processes. This dependence on the parallel structure is an obstacle considering the later discussed coupling of the MD simulation with the adaptive lattice-Boltzmann method. The adaptive LB simulation explained in Section 3.2 uses the P4EST library to manage the grid structure. The imposed linear order of the LB grid along the space-filling Morton curve yields an irregular domain decomposition. To avoid an additional communication step during the coupling, the linked cells grid has to be distributed in the same way. Due to the local space-filling structure, the resulting subdomains have no geometrical features that can be exploited by the communication or mapping algorithms. Thus, the existing MD simulation in the ESPRESSO software has to be adapted to work with P4EST and an almost arbitrary domain decomposition.

The linked cells algorithm implemented in ESPRESSO manages the cells and particles with pointer lists. All cells needed on a local process are stored in a continuous array. It consists of both local and ghost cells. These two kinds of cells are distinguished by separated pointer arrays. Each cell is a list of its contained particles with all their information of interest such as position, momentum, and applied force. The neighborhood information with pointers to the cell itself and to all 13 neighbors in the half-shell is associated to the array index of local cells. Additionally, there are several lists for communication storing cell pointers as origin or destination of the communication, the process index to interact with, and the potential particle shift due to periodic boundary conditions. Figure 4.6 is a simplified illustration of the lists and their connections.

This data structure makes no assumptions on the domain decomposition. Thus, the first step to integrate arbitrarily shaped domains into ESPRESSO is to fill the pointer lists with respect to the irregular decomposition. A forest consisting of regularly refined P4EST trees serves as basis for the MD grid. Dividing the absolute size of the simulation box by the maximum interaction range yields the number of cells in each direction. The refinement level of the trees is determined by finding the largest possible power of two dividing the number of cells in each dimension. The cell size can be enlarged such that the tree level increases as well in order to reduce the number of trees in the forest. Doing so, however, raises the computational effort per cell, because more interactions are computed. The distributed regular tree-based structure is created in parallel by P4EST routines.

The P4EST library provides information about the cell and process neighborhood as well as a minimal memory ghost layer. However, the linked cells algorithm requires a full halo of ghost cells around the domain. Depending on the domain's shape, it is possible to have multiple cells in the ghost layer that map to the same local cell of another or the same process. The

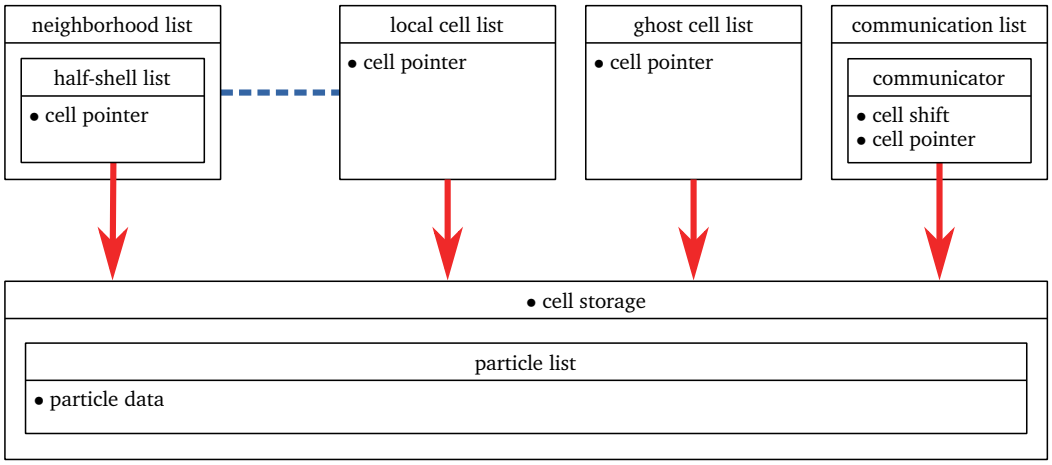


Figure 4.6: Schematic data structure for the linked cells algorithm used in ESPRESSO. Local cells and the neighborhood lists share the same enumeration. All structural information about the grid is stored in the half-shell lists. The connectivity to other processes is given by the communicators.

existing simulation updates the ghost cells in the corresponding communication step. The sending process shifts the particle positions in case the communication partner is connected over a periodic boundary. Thus, it is important to have these multiple ghost cells separated, because the included particles are seen at different physical positions for periodic copies although the original cell is the same. A surrounding shell around the local P4EST cells has to be constructed in order to create ghost cells for the linked cells algorithm. The difference between the minimal memory ghost layer used by P4EST and the full ghost shell is shown in Figure 4.7.

The neighborhood information provided by P4EST is not sufficient to represent the full ghost layer, since `p4est_ghost` does not include cells of the own process and no multiple copies. A self-defined ghost layer fitting the assumptions, however, is not compatible with the `p4est_mesh` structure as the neighborhood information changes. Thus, a new mesh structure for both local and ghost cells is defined to fulfill all needs of the linked cells algorithm. The existing data about cell and process neighborhood stored in the P4EST structures is used in order to compute the relevant cell information. The developed structure is explained in Table 4.1. The data structure is created in two steps. The first step, Algorithm 4.1, simply transfers all local P4EST cells. In the second step the ghost layer is computed and the neighborhood information is set as seen in Algorithm 4.2. This is done by searching the Cartesian coordinates of all full-shell neighbors for each local cell in the list of so far existing cells. A new ghost cell is added in case it does not exist yet. For newly added cells the `idx-rank` pair is counted in the existing list to keep track of multiple periodic copies of the same local P4EST quadrant. For ghost cells outside the simulation box the relative position is encoded in the `boundary`

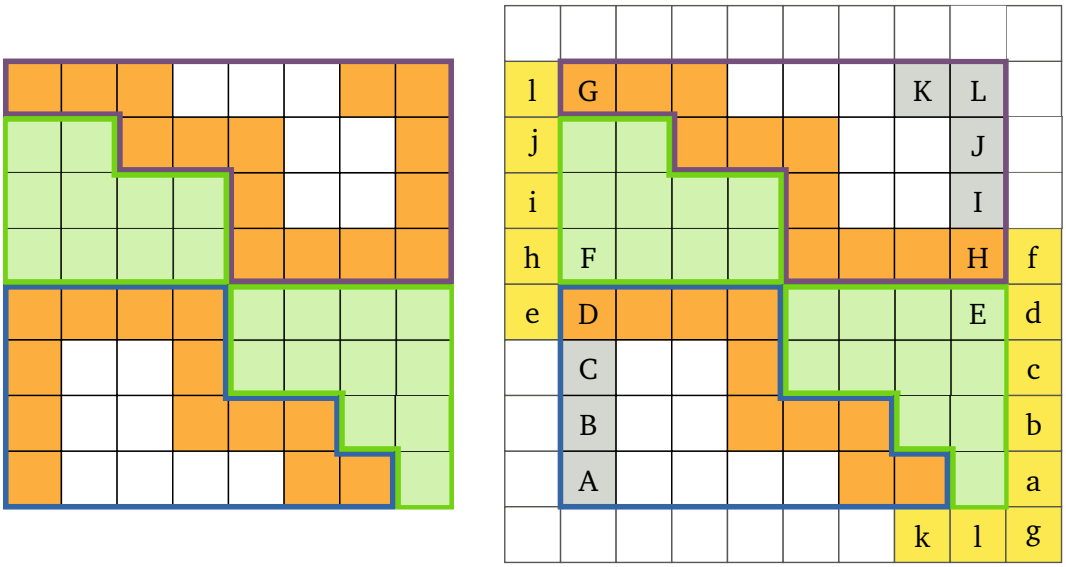


Figure 4.7: A regular fully periodic two-dimensional P4EST grid on three processes (blue, green, purple). Left: The orange cells are the minimal memory ghost layer provided by P4EST with respect to the green process. Right: The full ghost layer necessary for the linked cells algorithm is the union of yellow and orange cells. The additional yellow ghost cells are mapped to either local or original P4EST ghost cells. The mapping is indicated by matching letters. In the full ghost layer multiple copies (*D, G, H, L*) as well as mappings to the same process (*E, F*) exist.

type	name	description
`int`	`idx`	Index of the corresponding P4EST quadrant
`int`	`rank`	Index of the process for which the cell is inside the local domain
`int`	`shell`	Indicates the cell type (fully local, local mirror, ghost)
`int`	`boundary`	Encoded information about periodic boundaries for ghost cells
`int[26]`	`neighbors`	Indices of the full-shell neighborhood for local cells
`int[3]`	`coord`	Cartesian cell coordinate within the grid
`int`	`cnt`	Counter for multiple periodic copies of the same original cell

Table 4.1: The `cell_info` data structure for MD grid cells. The necessary geometric information of all cells allocated on a process is provided by this structure.

flag. The encoding for cells, both local and ghost, in the interior of the box is not used and thus set to a default value. The `boundary` value is later used to determine the particle shift due to periodicity. The loop over the 26 indices in Algorithm 4.2 uses the linear encoding for P4EST cell neighbors. I.e. faces are processed before edges and corners are accessed last. Thus, the first periodic copy of a cell, those having a `cnt` value of zero, have a minimal amount of folding directions with respect to their other periodic copies.

Algorithm 4.1 Copies the basic information of local P4EST quadrants to the MD grid structure.

```
1:  function CopyLocalCells(p4est forest, int level)
2:      grid ← {}
3:      for i ∈ range(forest.num_local_quadrants) do ▷ Copy information of local quadrants
4:          cell ∈ cell_info                ▷ cell uses the data structure defined in Table 4.1
5:          cell.idx ← i
6:          cell.rank ← getRank()
7:          cell.shell ← "local"
8:          cell.boundary ← −1
9:          cell.neighbors ← {}
10:         cell.coord ← ⌊p4est_quad_vertex(forest.local_quads[i]) · 2^level⌋
11:         cell.cnt ← 0
12:         grid.append(cell)                          ▷ Append copy to grid array
13:     end for
14:     return grid
15: end function
```

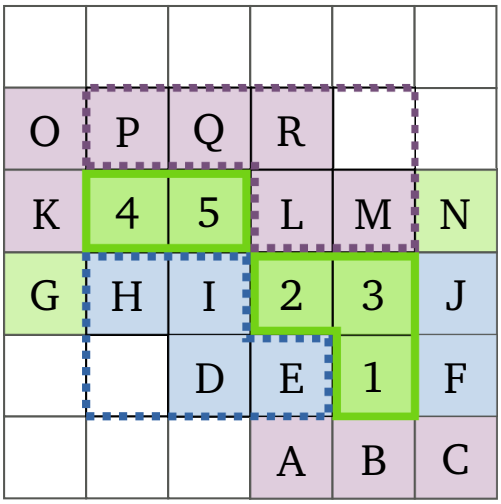
Another important issues with arbitrarily shaped domains is the missing structured information about neighboring processes used for communication. In a regular domain decomposition the communication is synchronous as the processes have a meaningful geometrical order. The irregular shaped surface for a P4EST based decomposition, however, makes asynchronous communication mandatory. A subdomain no longer has a fixed number of six neighboring processes over boundary faces. Additionally, the region assigned to a process can consist of more than one connected area. The MD simulation is based on two different types of communication steps, the ghost and the particle communication. The ghost communication updates the ghost layer of a process with the particle positions of neighboring processes, while the particle communication is invoked during the resorting step when particles leave the process local domain due to their motion. The particle migration, however, is discussed later in this section as it depends on a mapping of particles to local cells.

The ghost communication implemented in ESPRESSO is based on non-blocking MPI routines. It uses the communicator structure seen in Figure 4.6. A single send or receive ghost communication structure consists of the periodic particle shift and a list of either local mirror cells or ghost layer cells, respectively. Hence, a process has a send and receive communication structure for each neighboring process. Additionally, each occurrence of a process in the full

Algorithm 4.2 Creates the MD grid with all cell and neighborhood information for a given simulation box size and mesh width.

```
1: level ← max{l ∈ ℕ | 2^l ≤ ⌊box_l · mesh^-1⌋}                ▷ Determine regular tree level
2: brick ← ⌊box_l · 2^-level⌋                                  ▷ Compute number of trees in each dimension
3: forest ← P4EST_CREATE(brick, level)                         ▷ Create forest
4: grid ← COPYLOCALCELLS(forest, level)                        ▷ Copy data of local cells to cell_type array
5: for i ∈ range(forest.num_local_quadrants) do                ▷ Compute full ghost shell
6:     for n ∈ range(26) do                                    ▷ Check full neighborhood of all local cells
7:         cell ← grid[i]
8:         cell.coord ← cell.coord + NEIGHBOROFFSET(n)         ▷ Cell coordinate of neighbor
9:         cell.idx ← P4EST_NEIGHBOR(forest.local_quads[i], n).idx
10:        cell.rank ← P4EST_NEIGHBOR(forest.local_quads[i], n).rank
11:        if cell.coord ∉ range(⌊box_l · mesh^-1⌋) then       ▷ Cell outside simulation box
12:            cell.boundary ← n                               ▷ Set direction encoding
13:        end if
14:        cell.shell ← "ghost"
15:        found_idx ← size(grid) + 1
16:        for f ∈ range(grid) do                              ▷ Search for cell in existing grid and update counter
17:            if grid[f].idx = cell.idx and grid[f].rank = cell.rank then
18:                                                            ▷ Update respective periodic copy counter
19:                if cell.boundary < grid[f].boundary then
20:                    grid[f].cnt ← grid[f].cnt + 1
21:                else
22:                    cell.cnt ← cell.cnt + 1
23:                end if
24:            end if
25:            if grid[f].coord = cell.coord then              ▷ Found cell by its coordinates
26:                found_idx ← f
27:            end if
28:        end for
29:        if found_idx ∈ range(grid) then                     ▷ Cell already exists in grid
30:            grid[i].neighbors[n] ← found_idx                ▷ Update neighborhood information
31:            if grid[found_idx].shell = "ghost" then         ▷ Update encoding for ghost cells
32:                grid[i].shell ← "mirror"
33:                grid[found_idx].boundary ← min{grid[found_idx].boundary, n}
34:            end if
35:        else                                                ▷ Cell not found in grid
36:            grid.APPEND(cell)                               ▷ Append new ghost cell to grid
37:        end if
38:    end for
39: end for
```

ghost shell with a different particle shift needs its own communicator. The periodic shift is applied to particles before sending. Furthermore, the periodic shift is also used to compute a communication tag for MPI in order to distinguish between multiple communicators to the same process. Every sending communicator has a receiving associate on the same process as consequence of the bidirectional overlap in the mirror-ghost communication. This means, that for each sending communicator there is a receiving communicator for the same process with inverted particle shift. In special cases of the domain decomposition even communications to the same process exist due to periodic boundaries. Figure 4.8 visualizes the connection of ghost and mirror cells and the resulting communicators.



process	shift	send	receive
green	$l_X, 0$	4	N
	$-l_X, 0$	3	G
blue	$0, 0$	1, 2, 3, 4, 5	D, E, H, I
	$l_X, 0$		F, J
	$-l_X, 0$	1, 3	
purple	$0, 0$	2, 3, 4, 5	L, M, P, Q, R
	$l_X, 0$	4	
	$-l_X, 0$		K, O
	$0, l_Y$	1	
	$0, -l_Y$		A, B
	$-l_X, l_Y$	1	
	$l_X, -l_Y$		C

Figure 4.8: Exemplary ghost communicators (right) for the green process. The regular two-dimensional p4est grid (left) with box size (l_X, l_Y) is divided upon three process (blue, green, purple). Local cells (1 - 5) are sent to all processes appearing in the ghost layer. The ghost cells (A - R) are in the receiving communicators of the green process. For each sending communicator there is a receiving counterpart with inverted periodic shift.

The cells linked to a send and the corresponding receive list must have the same order as only the particle data without additional cell information is transmitted. Within the lists cells are ordered by the cell index property `idx`. It is assured by p4est that ghost quadrants have the same order as their corresponding local quadrants. Thus, mirror cells in a sending communicator have the same order as the respective ghost cells linked to the receiving communicator of the neighboring process. Each process can compute the send as well as the receive lists and the corresponding shifts without additional communication by using the grid structure constructed with Algorithm 4.2. Lists of receiving communicators are filled with pointers to ghost layer cells. Each ghost cell receives data from exactly one process. Thus,

the ghost cell's `rank` and `boundary` information is sufficient in order to assign the cell to the correct receiving communicator. The `boundary` flag is directly mapped to the periodic shift. However, mirror cells, that are linked to sending communicators, might appear more than once in ghost layers of other processes. Hence, it is possible for mirror cells to be part of multiple send lists, even to the same process. A mirror cell is added uniquely to each sending communicator with respect to the `rank-boundary` pair of all ghost cells in the mirror cell's full shell neighborhood. The shift for mirror cells is the inverted shift of the corresponding ghost cell for which it was added, because send and receive operations are opposing in their physical direction. Algorithm 4.3 is a simplified routine to create send and receive communicators.

Algorithm 4.3 Fills the send and receive communicator lists.

```
1:  function CreateCommunicators(cell_info[ ] grid)
2:      send ← communicator[NumProcesses][26]
3:      recv ← communicator[NumProcesses][26]
4:      num_cells ← size(grid)
5:      for cell_idx ∈ range(num_cells) do
6:          cell ← grid[cell_idx]
7:          if cell.shell = "ghost" then                ▷ Add ghost cell to receive communicator
8:              pos ← BinarySearch(grid[recv[cell.rank][cell.boundary].cells[•]].idx, cell.idx)
9:              recv[cell.rank][cell.boundary].cells.insert(pos, cell_idx)
10:             recv[cell.rank][cell.boundary].shift ← box_l · NeighborOffset(cell.boundary)
11:         end if
12:         if cell.shell = "mirror" then               ▷ Add mirror cell to send communicators
13:             for n ∈ range(26) do
14:                 ncell ← grid[cell.neighbor[n]]
15:                 if ncell.shell = "ghost" then       ▷ uniquely for each ghost in neighborhood
16:                     if cell ∉ send[ncell.rank][ncell.boundary].cells then
17:                         send[ncell.rank][ncell.boundary].cells.append(cell_idx)
18:                         shift ← box_l · NeighborOffset(ncell.boundary)
19:                         send[ncell.rank][ncell.boundary].shift ← −shift
20:                     end if
21:                 end if
22:             end for
23:         end if
24:     end for
25:     return send, recv
26: end function
```

The ghost communication of the MD simulation only updates the basic information of particles in neighboring cells to the process. This has to be done after every simulation step in which the position and momentum of particles change. The movement of particles, however, makes

it mandatory to reallocate the cell-membership of particles. Particles leaving the cell-local domain are migrated to the respective neighboring process. In the running simulation it is assumed that particles cannot move more than the mesh width within one time step. This prescribes a relation similar to the well-known CFL condition [CFL28] between the maximal particle velocity, the integration time step, and the mesh width of the grid. Exploiting this feature of the molecular dynamic simulation during the integration simplifies the process-local migration of particles between cells as well as the inter-process particle communication. As seen in Algorithm 4.4, particles are reallocated to either a new local cell or a particle send buffer in case they left the owning cell. Such particles belong to one of the 26 full-shell neighbors, due to the assumption stated above. The full-shell neighborhood is given by the

Algorithm 4.4 Updates the membership of all local particles based on their movement. Particles that moved into the ghost layer are attached to the send buffer of the neighboring process.

```
1:  function ResortParticles(cell[ ] cells, cell_info[ ] info)
2:      send_buf ← particle_list[NumNeighborProcesses]
3:      for (c, i) ∈ (cells, info) do                    ▷ Loop over all cells and particles
4:          for p ∈ c.particles do
5:              if MortonIndex(⌊p.coord · mesh^-1⌋) ≠ MortonIndex(i.coord) then
6:                  shell_idx ← RelativePosition(p.coord, c.mid_point)
7:                  new_cell ← i.neighbors[shell_idx]      ▷ Find neighbor by relative position
8:                  if info[new_cell].rank = ProcessID then                    ▷ Local cell
9:                      MoveParticle(p, cells[new_cell])
10:                 else                                                      ▷ Remote cell
11:                     MoveParticle(p, send_buf[info[new_cell].rank])
12:                 end if
13:             end if
14:         end for
15:     end for
16:     return send_buf
17: end function
```

`neighbors` field of the data structure computed with Algorithm 4.2. The relative position of the particle towards the cell center directly yields the correct neighbor cell. A local neighbor cell results in a reassignment of the particle, where for ghost cells the particle is moved to the send buffer of the respective neighboring process. After updating the membership of local particles, the inter-process particle migration is invoked. In case a particle leaves the global simulation box, it is folded by the sending process. All information necessary for folding is stored in the `boundary` attribute of the respective ghost cell. As for the ghost communication the particle migration has to be asynchronous, because of the missing structure for adjacent processes in the decomposition. For particles received this way the appropriate cells are found by Algorithm 4.5 in a single iteration over all local cells.

Algorithm 4.5 Inserting new particles into the correct local cell.

```
1:  function InsertParticles(particle_list recv_buf, cell[ ] cells)
2:      for c ∈ cells do                          ▷ Find owning cell for all particles in list
3:          for p ∈ recv_buf do
4:              if MortonIndex(⌊p.coord · mesh^{-1}⌋) = MortonIndex(i.coord) then
5:                  MoveParticle(p, c)
6:              end if
7:          end for
8:      end for
9:      if NotEmpty(recv_buf) then        ▷ If particles remain in list they are out of domain
10:         Error (Particles out of domain)
11:     end if
12: end function
```

Furthermore, particles are initially created on a single process which is not necessarily the process governing the respective physical domain, due to the manager-worker scheme of ESPResSo. The insertion algorithm 4.5 is also used after the global particle transfer that is needed for newly placed particles. The global communication is basically the same as the local particle update. It only varies in the amount of communication partners. The sending process of the global communication performs the particle folding resulting from periodic boundaries, since the receive routine of the local exchange can be reused this way. The routine to fill the send communication buffers, however, is different as it needs a mapping of particles to the governing process. The global space-filling order of the process-local domains is used for that purpose. Cartesian particle positions are mapped to a discrete three-dimensional cell index by dividing and rounding with respect to the mesh width. These three-dimensional coordinates are transferred into one-dimensional Morton indices using Algorithm 2.1. The p4est library provides an ordered list with the first quadrant on each process which is converted to global Morton indices as well using Algorithm 2.2. Each process-local domain is consecutive with respect to the space-filling curve. Thus, the governing process for a particle is found by a binary search in this list with respect to the particle's one-dimensional Morton index.

The global Morton index computed with Algorithm 2.1 and 2.2 for particle positions and cells with respect to the uniform refinement level of the MD grid is not necessarily equal to the cell index used by the p4est library, due to the grid's topology as already seen in Figure 2.5. Thus, the Morton index cannot directly be used to access cells in the general case. The process-local portion of the grid, however, is ordered with respect to z-curve index. This allows a mapping from linearized positions to local cells in $\mathcal{O}(\log N)$ for N local cells by using a binary search. However, for some processes, mostly those in the interior of the simulation box, the global space-filling curve fully covers the subdomain without leaving it. Hence, the global Morton index only differs by a constant offset from the enumeration used by p4est. Processes with this property are characterized by a difference in the global Morton index

to the successive process equal to the number of local cells. Exploiting this feature yields a position to cell mapping in $\mathcal{O}(1)$, as the cell index and the global curve index only differ by an additive constant. The local index for cells on those processes used to access them in the storage is given by subtracting the global index of the respective first cell.

So far numerical rounding errors caused by the particle motion, periodic shifts, and the integer linearization of the Morton index were neglected. In a general setting, however, these numerical errors need to be considered, since they cause the software to malfunction. A big issue is the treatment of particles directly located on cell boundaries. Particles are assigned to cells by integer rounding. Thus, particles on the lower boundary are still mapped to the cell where those on the upper boundary are located in the respective adjacent cell. In principle, this is no problem as long as the particle stays in the cell-local subdomain. On interprocess boundaries, however, this behavior might cause additional communication, because the particle migrates back and forth between the processes due to numerical fluctuations during the particle motion. Furthermore, particles directly located on the simulation box boundary are wrapped to the respective other boundary in a periodic setting. Particles on the upper boundary of the simulation box are mapped to cell indices outside of the valid domain. This issue is not particularly caused by the P4EST implementation of the MD algorithm. In the original ESPRESSO code the process-local subdomain is increased by a small factor for error rounding (i.e. $\varepsilon \cdot \mathbf{box_l}$, with $\varepsilon = 10^{-14}$) such that it minimally overlaps with the neighboring processes. This causes particles to stay on a process although they are outside but close to the local domain. In the same way particles are accepted by processes and placed in local cells after a particle migration. This scheme of extended domains reduces additional communications based on numerical fluctuations, because a particle is placed in the interior of the neighboring process instead on its boundary as soon as it leaves the extended area. Extending the process-local domain is quite simple in case of the regular domain decomposition of the original ESPRESSO code. The P4EST based decomposition, however, makes this method more cumbersome, due to the irregular shaped domains. Instead of extending the local domain area, the particle position is shifted by the rounding factor in all directions. In other words, the overlap of two volumes needs to be checked to determine if a position is inside the extended domain. It is sufficient to shift only in diagonal directions towards the corners, since these eight positions mark the corners of the tested subvolume. In total the nine (the original position plus the eight shifted) are mapped to the local cells by the method described above. If none of the values is in the local domain, the position is outside of the extended area. In case the original position is inside a local cell, this cell is returned. Only if the original position is outside of all local cells the shifted values are considered.

CHAPTER 5

Particle-Fluid Coupling

Various applications and researches are based on the behavior and interaction of solvent particles or polymers. For classical molecular dynamics both, the polymer and the fluid, are simulated as particles [DK93, KG90, PR92], but one is mainly interested in the behavior of the solvent structure. Additionally, this often comes along with high computational costs as the number of particles used for the solvent has to be much larger than the amount of actual polymer particles. Representing the fluid by a continuum assumption seems convenient for systems with a huge gap in the physical scale from the fluid to the interacting molecules. Ahlrichs and Dünweg [AD99] proposed a scheme for the viscous coupling of molecular dynamics with the lattice-Boltzmann method. Moving an object through a flow results in a frictional force between the fluid and the particle depending on the relative velocity. Coupling the particle motion with the flow thus requires the interpolated flow velocity at the position of each particle. Similar to Stokes formula for a moving sphere in a viscous fluid the resulting force on the discretized fluid values and the moving object due to friction is given by

$$\mathbf{F}_{fl} = -\zeta(\mathbf{v}_p(t) - \mathbf{u}(\mathbf{x}_p(t), t)) \tag{5.1}$$

where ζ is a proportionality coefficient and $\mathbf{u}$ is the flow speed. The velocity and position of the particle is given by $\mathbf{v}_p$ and $\mathbf{x}_p$, respectively. Stochastic terms, as for Brownian motion, have to be added to both fluid and particles. Extending Equation 5.1 yields

$$\mathbf{F}_{fl} = -\zeta(\mathbf{v}_p(t) - \mathbf{u}(\mathbf{x}_p(t), t)) + \mathbf{f}_{st} \tag{5.2}$$

with the stochastic force $\mathbf{f}_{st}$ with zero mean and the relation

$$\left\langle f_{st,\alpha}(t) f_{st,\beta}(t') \right\rangle = \delta(t - t') 2\delta_{\alpha\beta} k_B T \zeta \ . \tag{5.3}$$

The flow speed $\mathbf{u}$ needs to be interpolated using the values at the discrete Boltzmann lattice points. For particles in a fully regular grid or in locally regular parts of an adaptive grid the eight nearest fluid values with respect to the particle position are combined using the trilinear interpolation

$$\mathbf{u} = \sum_{i,j,k=0}^{1} \alpha_i \beta_j \gamma_k \ \mathbf{c}_{i,j,k} \tag{5.4}$$

visualized in Figure 5.1. The weights $\alpha_i, \beta_j, \gamma_k$ are in the interval $[0, 1]$ and satisfy the condition $\lambda_1 = 1 - \lambda_0$, respectively. The lattice-Boltzmann method stores the fluid quantities in its

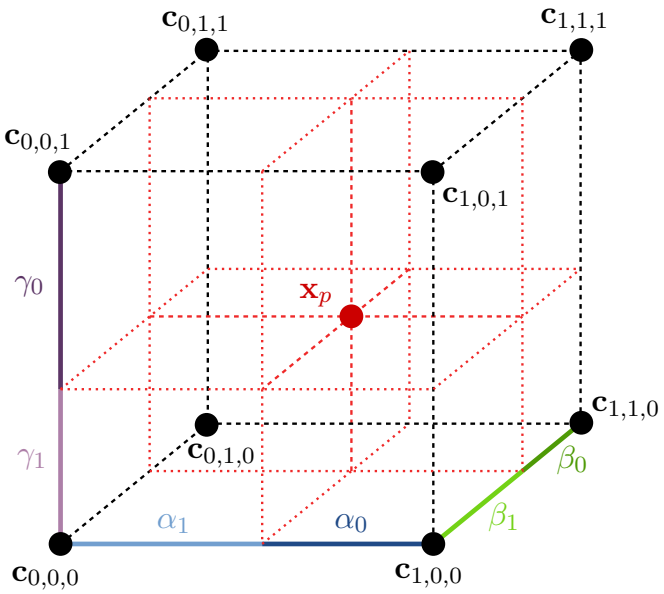


Figure 5.1: Trilinear interpolation in a regular Cartesian lattice. The six weights $\alpha_i, \beta_j, \gamma_k$ are the normalized distances in Cartesian direction from the given position $\mathbf{x}_p$ to the grid points associated to the values $\mathbf{c}_{i,j,k}$.

cells. An obvious and easy point-wise representation of the data is given by linking the values to the cell centers. Hence, for the linearized representation of the grid along the z-curve a mapping from Cartesian coordinates to the linear Morton index is needed. The existing ordered structure of the grid can be exploited for this purpose. The three-dimensional coordinates are simply transferred to a one-dimensional Morton index using Algorithm 2.2 with respect to the global grid level. Afterwards, the cell containing the given position is accessed by a binary search in the local tree data. Algorithm 5.1 is used as comparator for the particle's coordinate and the LB cells. The remaining seven cells for interpolation can be accessed using the relative position of the particle to the cell center and the cell's neighborhood information.

Algorithm 5.1 Comparator to check if a Cartesian coordinate is located inside a P4EST cell.

```
1: function ParticleInCell(quad cell, float[3] pos)
2:     ipos ← ⌊pos · 2^cell.level⌋          ▷ Element wise integer rounding on cell level
3:     if GlobalQuadIndex(cell, cell.level) = MortonIndex(ipos) then
4:         return 1
5:     end if
6:     return 0
7: end function
```

After determining the suboctant of the cell containing the position and the corresponding weights $\alpha_i, \beta_j, \gamma_k$ used in (5.4), the seven neighbor indices in P4EST notation can be looked up using Table 5.1. Suboctants are also ordered along the Morton z-curve. Algorithm 5.2 shows the interpolation in the local domain and the computation of the trilinear weights. The interpolation for adaptive, thus more complex, grids is explained in Section 5.3.

suboctant	**index**	**P4EST neighbor indices**						
left, front, bottom	0	f0	f2	f4	e0	e4	e8	c0
right, front, bottom	1	f1	f2	f4	e0	e5	e9	c1
left, back, bottom	2	f0	f3	f4	e1	e4	e10	c2
right, back, bottom	3	f1	f3	f4	e1	e5	e11	c3
left, front, top	4	f0	f2	f5	e2	e6	e8	c4
right, front, top	5	f1	f2	f5	e2	e7	e9	c5
left, back, top	6	f0	f3	f5	e3	e6	e10	c6
right, back, top	7	f1	f3	f5	e3	e7	e11	c7

Table 5.1: Lookup table of P4EST neighbor indices for a given suboctant index.

Algorithm 5.2 Interpolation on local cells of a regular P4EST grid

```
1:  function LocalInterpolate(float[3] pos, p4est lb_grid)
2:      base_quad ← ∅
3:      for q ∈ lb_grid.local_quads do          ▷ Find quadrant containing the position
4:          if ParticleInCell(q, pos) then
5:              base_quad ← q
6:          end if
7:      end for
8:      if base_quad = ∅ then                   ▷ Position not in local domain
9:          return ∅, ∅
10:     end if
11:     rel_subquad ← mod(MortonIndex(⌊pos · 2^(base_quad.level+1)⌋), 8)   ▷ Relative index
12:     for d ∈ range(3) do                     ▷ Compute trilinear weights
13:         αβγ[d][0] ← 2^(base_quad.level) · |base_quad.mid_point[d] − pos[d]|
14:         αβγ[d][1] ← 1 − αβγ[d][0]
15:     end for
16:     cells[0] ← base_quad
17:     for n ∈ range(7) do          ▷ Gather neighboring cells using the lookup table 5.1
18:         cells[n + 1] ← p4est_neighbor(base_quad, LookUp[rel_subquad][n])
19:     end for
20:     return cells, αβγ
21: end function
```

Due to Newton's third law of motion the force (5.2) applies to both the particle and the fluid. The force (5.2) is simply accumulated with other forces affecting the particle and the overall sum is applied in the integration step of the molecular dynamic simulation. The reactional force towards the fluid is passed to the lattice as incremental momentum density $\Delta\mathbf{j}$ within one MD time step. Considering the normalization of the lattice-Boltzmann quantities with respect to the LB time step and the finest mesh width it reads as

$$\Delta\mathbf{j} = -\mathbf{F}_{fl}\ \frac{h_{\max}^3}{h^3}\ \Delta t_{MD}\ \frac{\tau_{LB}}{h_{\max}} \tag{5.5}$$

where h is the lattice mesh width of the current cell and $\tau_{LB}/h_{\max}$ the normalization factor to convert the force to LB units (cf. [AD99] Equation (12)). The momentum density in the dimensionless formulation of the Boltzmann equation is scaled by the reference volume $h_{\max}^{-3}$. For a fully regular grid h equals $h_{\max}$ and the dimensionless force density $-\mathbf{F}_{fl}\ h_{\max}^3\ h^{-3}$ becomes $-\mathbf{F}_{fl}$. In an adaptive setting, however, the actual mesh width h might be larger than the minimal mesh size $h_{\max}$ by a power of two. Equation (5.5) transforms to

$$\Delta\mathbf{j} = -\mathbf{F}_{fl}\ 2^{-3\Delta l}\ \Delta t_{MD}\ \frac{\tau_{LB}}{h_{\max}} \tag{5.6}$$

using the level difference Δl between the current cell level and the maximal allowed level. The change in the momentum density Δj is split among the eight grid cells used for interpolation by reusing the same weights $\alpha_i, \beta_j, \gamma_k$

$$\Delta\mathbf{j}_{i,j,k} = \alpha_i\beta_j\gamma_k\ 2^{-3\Delta l_{i,j,k}} \left(-\mathbf{F}_{fl}\ \Delta t_{MD}\ \frac{\tau_{LB}}{h_{\max}}\right) \tag{5.7}$$

where the part in brackets is independent from the actual fluid cell.

On a single processing node particles in periodic ghost cells don't have to be coupled, because the neighborhood information of the LB grid also wraps around periodic boundaries as seen in Figure 5.2. Hence, all interactions of particles in MD ghost layers with the fluid are already processed by the splitting (5.7) of the original particle. Coupling particles with the lattice-Boltzmann method on several processes, however, has effects on the interpolation and splitting as well as on the interaction of particles in the ghost layer. As constructed in Section 4.3, the ghost layer of the MD simulation forms a full shell around the local domain. Hence, it is possible to have more than one ghost cell as image of an internal cell. The P4EST structure for the lattice-Boltzmann method, on the other hand, uses a minimal-memory set of halo cells without periodic duplicates. This difference in the ghost layer of the MD and LB algorithms has a severe influence on the interaction of particles placed in ghost cells with the fluid. Particles in periodic ghost cells have Cartesian coordinates outside the simulation box. To locate the correct LB halo cell for such particles their position has to be folded inside the allowed domain. This maps the external ghost cell back to its original position which is either a local cell or an actual ghost cell. Thus, processes with more than one copy of a MD cell, whether it is a local, inner ghost or a periodic ghost cell, would compute the particle-fluid interactions multiple times due to the same reason visualized for a single process in Figure 5.2. For a

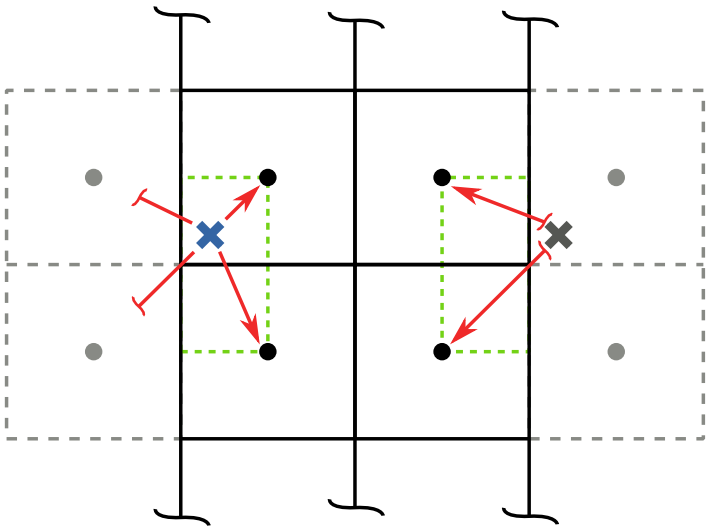


Figure 5.2: Coupling at periodic boundaries. The interpolation area (dashed green) for the particle position marked by the blue cross near the vertical periodic boundary wraps around the domain. The weighted forces of this particle also represent those for its copy (gray mark) in the ghost layer (dashed gray cells).

correct multi-threaded coupling only the particles of one periodic copy for each MD cell must be processed. To avoid unnecessary folding, local cells and inner ghost cells are preferred over copies in the periodic ghost layer. It is not necessary to search all local data to map positions of ghost layer particles to LB cells. The P4EST library provides separated lists for ghost and mirror cells that can be used for this purpose. Mirror cells, those cells that appear as ghost in other processes, are ordinary local LB cells. The interpolation and splitting weights are computed in the same way as for the local particle mapping. For ghost cells, however, the neighborhood information is not provided by P4EST. After determining the suboctant of the ghost cell containing the position, the global space-filling curve indices of the seven missing interpolation positions are computed. These indices are used to find the corresponding cells in the ghost or mirror cell list. For particles located near the outer boundary of ghost cells not all seven neighbor cells are stored in the process local data. These particles are ignored, since they are processed as particles with purely local effects on another process. The interpolation on ghost cells is shown in Algorithm 5.3.

The lattice-Boltzmann fluid simulation and molecular dynamics operate on two different time steps for integration. The LB time step τ_{LB} is forced to be larger or equal to the MD time step Δt_{MD} due to stability reasons. In the dimensionless formulation of the lattice-Boltzmann method the parameter τ_{LB} is used as reference. Thus, it is linked to the spatial quantity $h_{\max}$ on the finest possible level. The level subcycling scheme states that during the integration LB cells on the finest level with the mesh width $h_{\max}$ are processed every τ_{LB}. Cells on coarser levels are only integrated every $2^{\Delta l}$-th step where Δl is the level difference compared to the

Algorithm 5.3 Interpolation on ghost cells of a regular P4EST grid

```
1:  function GhostInterpolate(float[3] pos, p4est lb_grid)
2:      base_quad ← ∅
3:      for q ∈ lb_grid.ghost_quads do          ▷ Find ghost quadrant containing the position
4:          if ParticleInCell(q, pos) then
5:              base_quad ← q
6:          end if
7:      end for
8:      if base_quad = ∅ then                   ▷ Position not in ghost layer
9:          return ∅, ∅
10:     end if
11:     rel_subquad ← mod(MortonIndex(⌊pos · 2^(base_quad.level+1)⌋), 8)   ▷ Relative index
12:     for d ∈ range(3) do                     ▷ Compute trilinear weights
13:         αβγ[d][0] ← 2^(base_quad.level) · |base_quad.mid_point[d] − pos[d]|
14:         αβγ[d][1] ← 1 − αβγ[d][0]
15:     end for
16:     cells[0] ← base_quad
17:     for n ∈ range(7) do                     ▷ Gather neighboring cells using the lookup table 5.1
18:         shift_pos ← pos + 2^(−base_quad.level) · NeighborOffset(LookUp[rel_subquad][n])
19:         cells[n + 1] ← ∅
20:         for q ∈ lb_grid.ghost_quads ∪ lb_grid.mirror_quads do
21:             if ParticleInCell(q, shift_pos) then   ▷ Cell in ghost or mirror found
22:                 cells[n + 1] ← q
23:             end if
24:         end for
25:         if cells[n + 1] = ∅ then            ▷ Position not close enough to local domain
26:             return ∅, ∅
27:         end if
28:     end for
29:     return cells, αβγ
30: end function
```

finest level. The coupling has to be computed every LB integration step even if the involved LB cells are on higher levels, because the particle interaction on the fluid is scaled with respect to these reference quantities as seen in Equation (5.7). The level based factor of the splitting scheme (5.7) changes by applying the incremental momentum density only to cells that are processed with respect to the subcycling. This results in

$$\Delta \mathbf{j}_{i,j,k} = \alpha_i \beta_j \gamma_k \, 2^{-2\Delta l_{i,j,k}} \left(-\mathbf{F}_{fl} \, \Delta t_{MD} \, \frac{\tau_{LB}}{h_{\max}} \right) . \tag{5.8}$$

Equation (5.8) couples particle forces only with cells that are integrated during the same time step. This has a numerical influence on the solution, because particles are able to change

their position and momentum within one macro time step of coarser cells. Additionally, the streaming step on coarser cells of the adaptive lattice-Boltzmann method is delayed compared to the regular LB scheme as seen in Figure 5.3. Thus, the fluid population seen by the coupling algorithm and the interpolation in particular is older by $\mathcal{O}(\Delta t)$ compared to the regular LB scheme with Δt as time step of the coarse level, because the fluid populations are only updated after the respective streaming step.

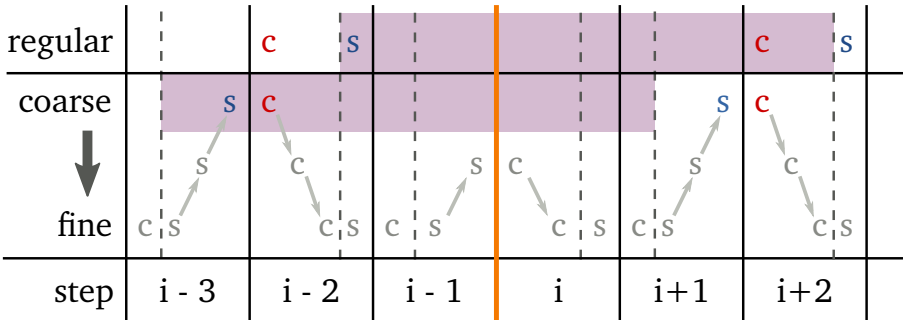


Figure 5.3: Comparison of the regular and adaptive lattice-Boltzmann method on a common coarse grid level. Collision (*c*) and streaming (*s*) on finer levels of the adaptive LB method are grayed out as they are skipped on not fully refined regular grids. The lifespan of the fluid populations used for interpolation is highlighted in purple. The coupling is performed between two LB steps. Thus, the i-th coupling marked by the orange line is executed before the i-th fluid step. In the adaptive case the fluid populations are older by $\mathcal{O}(\Delta t)$ compared to the regular LB method with Δt as time step on the coarse level, because the regular streaming in step $i-2$ corresponds to the adaptive streaming in step $i+1$.

5.1 Coupling for Similar Grids Using the Finest Common Tree

Distributing the simulation on several processes leads to a domain decomposition of both the linked cells grid for molecular dynamics and the lattice discretizing the Boltzmann equation. The domain decomposition imposed by using p4est and space-filling curves splits the linearized grids (equally or weighted) among all processes. Two different grids, even regular grids on different levels, thus do not share the same subdomains in general. Figure 5.4 shows this in a simple two-dimensional case for two regular grids on different levels split in three almost equal subdomains. Due to the interpolation scheme (5.4), the frictional force $\mathbf{F}_{fl}$ as in Equation (5.2) cannot be computed without communication, since there are particles in cells that are not distributed to the same process with respect to the overlapping fluid cell. Hence, the local subdomains for both, the particle and fluid simulation, must be aligned in order to couple them in a meaningful and minimally invasive manner.

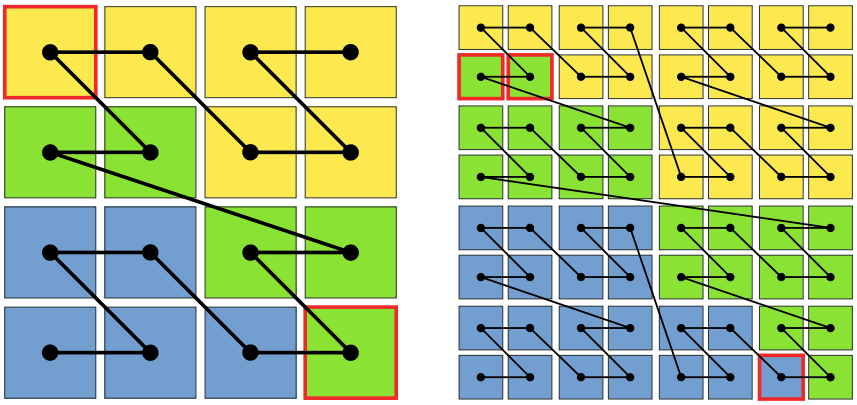


Figure 5.4: Equal distribution (up to integer rounding) of a grid on level two (left) and on level three (right) among three processes along the Morton curve. The cell membership is colored in blue, green, and yellow. Cells highlighted in red partly belong to a different process compared to the respective other grid.

In case the two grids of the MD and the LB simulation share a common basis in the sense of the same set of root nodes, the above prescribed mismatch in the distribution can be solved by repartitioning the grid. For P4EST based grids this means, that both the LB and the MD forest are at least created upon the same `p4est_connectivity`. As long as one of both grids is on a lower level for all cells it is easy to just move the problematic cells of the finer grid to other processes. This can be done fully parallel, since the already distributed finer grid only has to find the appropriate processes for each of its local cells with respect to the coarser grid. It is sufficient to know the global Morton curve index of the first cell for each subdomain of the coarser grid. Local cells for the finer grid can be mapped by using at least the highest level of the coarser grid as reference for Algorithm 2.2. For N local cells and P processes this mapping as seen in Algorithm 5.4 is in $\mathcal{O}(N + P)$, since both lists are sorted. The P4EST library provides methods for weighted and explicit repartitioning, but both require a global communication step in advance.

Algorithm 5.4 aligns the local process boundaries of two grids under the assumption that the global minimal level of one grid is equal to or greater than the global maximal level of the other. This assumption is always fulfilled in case of regular grids. For adaptive refined LB grids, however, it is possible to have finer as well as coarser cells compared to the regular MD grid. To align the boundaries the grid needs to be split along the respective coarser cells. The easiest forest to use as reference is the common basis of both LB and MD. But this is not the best solution considering freedom in load balancing as the base forest usually has very few octants. The best possible solution is the *finest common tree* gained by intersecting the MD and LB forests. It can be used as reference grid for Algorithm 5.4 as it satisfies the assumptions

Algorithm 5.4 Repartitions the finer forest fgrid to be aligned with the partitioning of the coarser forest cgrid. Both forests must be based on the same `p4est_connectivity`.

```
1:  procedure REPARTITION(p4est cgrid, p4est fgrid)
2:      ref_level ← cgrid.finest_level_global
3:      for p ∈ range(P) do          ▷ Initialize array with index of first quadrant per process
4:          num_per_proc[p] ← 0
5:          global_first_idx[p] ← GLOBALQUADIDX(cgrid.first[p], ref_level)
6:      end for
7:      p ← 0
8:      n ← 0
9:      while n ∈ range(fgrid.num_local_quadrants) do
10:         if p + 1 ∈ range(P) then      ▷ Loop either over cells or processes and count cells
11:             quad_idx ← GLOBALQUADIDX(fgrid.local_quads[n], ref_level)
12:             if global_first_idx[p + 1] < quad_idx then
13:                 num_per_proc[p] ← num_per_proc[p] + 1
14:                 n ← n + 1
15:             else
16:                 p ← p + 1
17:             end if
18:         else                          ▷ Remaining cells belong to last process
19:             num_per_proc[p] ← num_per_proc[p] + 1
20:         end if
21:     end while
22:     ALLGATHERSUM(num_per_proc)                    ▷ Global communication step
23:     P4EST_PARTITION(fgrid, num_per_proc)          ▷ Repartition finer grid
24: end procedure
```

with respect to both other grids. Figure 5.5 shows the finest common tree for two arbitrarily refined two-dimensional trees.

The finest common tree can be constructed in an incremental and a decremental way. The incremental method starts with the root and recursively refines the nodes if they are inner nodes for both existing trees. The decremental method starts with a copy of one tree and recursively coarsens the tree as long as the nodes do not exist in the respective other tree. Although both methods result in the same tree, they are not equally suited for parallel distributed data. A general issue concerning parallelism is the distribution of the trees among the processes, because local information of all trees is needed to coarsen or refine. The decremental construction starts as copy of one grid and thus only needs the local cell data of the other grid, whereas the incremental method depends on both trees. Furthermore, the incremental tree has very few nodes to begin with, so it needs to be repartitioned every step to keep a good load balance. Algorithm 5.5 is a fully parallel version of the decremental method

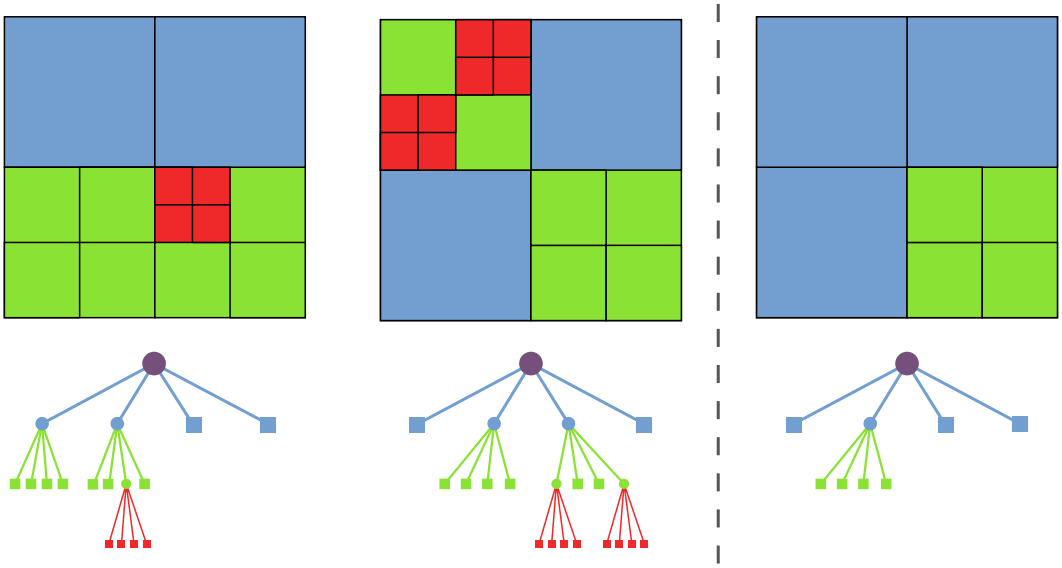


Figure 5.5: Left of the dashed line: Two arbitrarily refined grids (top) and its corresponding trees (bottom). Intersecting these two trees results in the finest common tree (bottom) on the right. Tree nodes and grid cells on the same level are highlighted in the same color. Leaf nodes that represent actual grid cells are shown as squares. Note that these grids are for illustration purposes only as they are not conforming with the 2:1 balancing.

for constructing the finest common tree using the P4EST library. To have access to the local tree information for coarsening the copy is repartitioned using Algorithm 5.4. The assumptions, however, are not satisfied and thus the domain boundaries are not aligned. Nevertheless, the repartitioned copy fits all requirements for coarsening: a) In case a node of the copy is finer than in the reference tree, all quadrants with the same parent as this node are copied to the correct process to be refined later; b) In case a node is equal or coarser than in the reference tree, it is copied to a process such that it overlaps with the other tree's node, but it must not be coarsened as it is already on the same or a lower level. After constructing the finest common tree from the MD and LB grid it can be used to repartition both grids using Algorithm 5.4. The explained method to construct the finest common tree does not use the fact the MD grid is always regular which implies that all cells are on the same level. Algorithm 5.5 uses the grid's level in the callback function. Thus, by copying the LB structure and recursively coarsen it the loop over the quadrants of the MD forest can be replaced by a simple check against its constant level.

Algorithm 5.5 Constructs the finest common tree with respect to the arbitrary forests t1 and t2 that share the same `p4est_connectivity`.

```
1:  function FinestCommonTree(p4est t1, p4est t2)
2:      fct ← p4est_copy(t2)
3:      Repartition(t1, fct)              ▷ Repartition copy to have access to local data of t1
4:      fct.user_ptr ← t1
5:      p4est_coarsen(fct, CoarsenCallback)                    ▷ Recursive coarsening
6:      p4est_partition(fct)                                    ▷ Repartition equally
7:      return fct
8:  end function
9:
10: function CoarsenCallback(p4est forest, int tree, quad[ ] quads)
11:     ref_level ← forest.finest_level_global
12:     qidx ← GlobalQuadIdx(quads[0], ref_level)
13:     for q ∈ forest.user_ptr.trees[tree].quads do
14:         if GlobalQuadIdx(q, ref_level) = qidx and q.level >= quads[0].level then
15:             return 0                          ▷ fct is on same or lower level for this node
16:         end if
17:     end for
18:     return 1                        ▷ Node not found in other tree with at least this level
19: end function
```

5.2 Coupling for Grids with Arbitrary Mesh Widths

The mesh width of the particle simulation is given in dependency of the cut-off radius of the short-ranged interaction potential. Whereas, the lattice discretizing the Boltzmann equation, especially in case of adaptive refinement, represents only the spatial resolution. For a general setting this implies that the mesh widths of the particle and the flow simulation are not connected by a factor of the power of two. In general, the cells of the MD and the LB grid are not aligned at all. Considering a domain decomposition with respect to one of both grids, this leads to process domain boundaries that intersect the respective other grid cells as illustrated in Figure 5.6. In general, a cell can be divided among more than two processes. To couple both simulations the particles stored in the MD cells must be linked to the fluid velocities. But having an intersecting process domain boundary is a problem, since it is not clear to which process the divided cell should belong. Furthermore, there might exist particles that are out of the process-local domain with respect to the LB grid.

The nature of the linked cells algorithm always yields a regular grid, whereas the structure for the adaptive Boltzmann lattice is much more complex. Thus, it seems convenient to let the process domain boundaries be prescribed by the fluid cells and have the regular MD grid be intersected. This even comes along with another advantage. On the contrary to

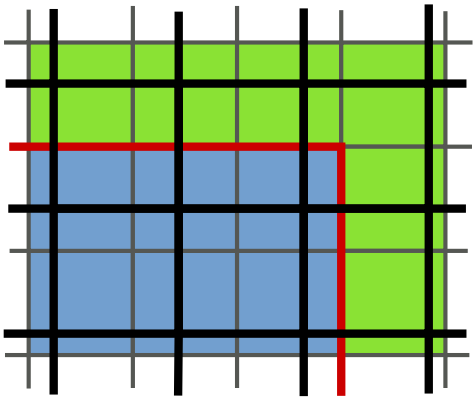


Figure 5.6: Domain decomposition intersecting cells of a grid with an arbitrary mesh width. The domain is divided along the gray grid into the green and the blue area. The red domain boundary intersects cells of the black grid that has a different mesh width.

the lattice-Boltzmann simulation, where the computational effort is linked to the cells, the workload of the MD simulation is directly coupled to the amount of particles per cell. Hence, it can be divided among all involved processes. In order to achieve this, the cells for the particle simulation have to be further distinguished. In addition to the already existing local and ghost cells those being intersected by the process domain boundary are labeled as *semi*-cells. As only a part, the overlapping area with the process-local domain, of a semi-cell is computed by a process, it can be split into a *semi-local* and a *semi-ghost* cell. Figure 5.7 shows local, semi, and ghost cells for the domain boundary cutting and being aligned with the grid. The process-local domain and both the semi-local and semi-ghost areas are highlighted. Semi-local and semi-ghost cells govern the same region as they are basically the same cell with two different algorithmic purposes. However, semi-local cells only store particles located in the process-local part of the cell. The local neighborhood information and the structure of the cell's half-shell in particular changes due to semi-cells. Both need to be accessed by adjacent cells in order to compute all interaction forces. Algorithmically, semi-cells are treated as their respective regular equivalents. Only the migration of particles between cells and processes needs to be adapted. Particles contained by a semi-local cell that left the process-local domain have to be moved to the corresponding process with respect to their position, even if they are still inside the cell's region. The semi-ghost cells are then filled and overwritten within the ghost-communication step of the algorithm. Thus, in case a semi-cell overlaps more than two subdomains, each involved communication partner needs its own semi-ghost cell to guarantee minimal changes of the existing algorithms.

The simulation software ESPRESSO uses lists of pointers for the neighborhood information. This makes it easy to extend the code by semi-cells. Only the functions for creating the grid, filling the pointer lists, and migrating particles need to be changed. The computation of the interactive force and the ghost communication work as they are, as long as semi-cells are

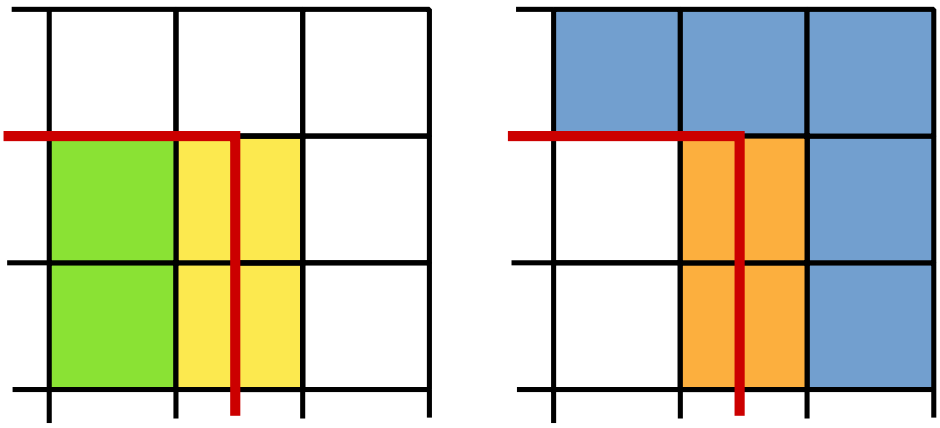


Figure 5.7: Domain boundary intersecting and being aligned with the grid. The local and semi-local cells on the left are highlighted in green and yellow, respectively. They are computed by the local process. The ghost and semi-ghost cells on the right are filled during the communication step. Regular ghost cells are shown in blue, where semi-ghost cells are colored orange.

marked as regular local or ghost cells, respectively. The neighborhood list for a semi-cell contains pointers to its semi-local, all its semi-ghost entities, and to adjacent half-shell cells. For a neighboring semi-cell in the half-shell pointers to all entities have to be listed as well. This results in lists of varying length with at least 14 entries as before. However, it is assured that all pair-interactions are still computed once, because only particles in local (semi-local and real local) cells are processed and each semi-local cell appears as a semi-ghost in all neighboring processes.

5.3 Coupling for Adaptive Lattice-Boltzmann Grids

The resolution and accuracy of the fluid simulation is improved by refining the LB grid. To avoid the exponential growth in the complexity that comes along with a regular refinement, one can adaptively refine around areas of interest, e.g. physical boundaries inside the fluid. An adaptive grid has cells of different sizes, but we restrict it by imposing a 2:1 balancing to reduce the algorithm's complexity. Thus, neighboring cells are only allowed to have a level difference of at most one. The particle-fluid coupling consisting of the interpolation (5.4) and the splitting (5.7) requires local neighborhood information for the cells. Particles that are placed in an adaptive refined part of the LB grid interact with neighboring cells with the same, half, or twice the mesh width compared to its containing cell. Hence, the number of cells used for interpolation and splitting has no fixed number as for regular grids. A big issue with the adaptive lattice-Boltzmann method is the question on how to interpolate and compute weights

for non-regular grids, since the regular trilinear interpolation scheme cannot be applied. Figure 5.8 points out the problem for a less complex two-dimensional case. The resulting

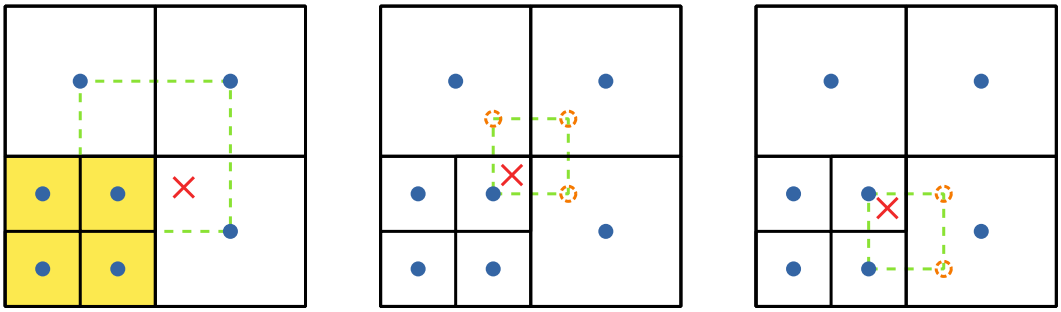


Figure 5.8: On adaptively refined grids the classical linear interpolation cannot be used. The left grid has finer cells with respect to the position marked with the red cross. It is not clear which of the yellow cells should have an influence on the interpolated value. For the interpolation with coarser cells as for the other both grids there are missing points (orange circles) in the interpolation area (green dashed line).

interpolation should be continuous over cell boundaries to avoid additional numerical errors. Furthermore, the weights should be consistent with the trilinear interpolation in the regular case, such that the method yields the same results. The P4EST mesh structure grants access to all directly adjacent cells. For neighbors over a face this can either be one or four cells in a fully periodic simulation box. Edges are connected to none, one, or two cells and corners only map to at most one cell. If no cell is returned as neighbor over an edge or corner, there exists a double sized cell over a face covering that area, in case the `enforce_same_size` flag of the P4EST routine is set. The number of cells needed for interpolation on adaptive grids is in the range from five to 20. To recall, the regular case always uses eight grid nodes and combines their weights by multiplying three out of six one-directional weights. In the adaptive setting simply combining the weights is not possible, due to the variety in the number of used cells. Thus, the interpolation scheme (5.4) becomes

$$\mathbf{u} = \sum_{m=0}^{|\{\mathbf{c}_n\}|} \lambda_m \, \mathbf{c}_m \ , \tag{5.9}$$

where $\{\mathbf{c}_n\}$ is the set of all neighboring cells regardless of their size. The coefficient λ_m is the weight corresponding to $\mathbf{c}_m$. It reads as

$$\lambda_m = \hat{\lambda}_m \, \alpha_i \, \beta_j \, \gamma_k \ , \tag{5.10}$$

with i, j, k given by the direction in which $\mathbf{c}_m$ is found. The adaptive splitting scheme uses the same weights and cells. The scaling factor $\hat{\lambda}_m$ for half sized neighbors is computed by dividing the combined regular weights by the number of cells in this direction. This totals to an

additional factor of 0.25, 0.5, or 1.0 for half sized cells over faces, edges, or corners, respectively. Weights for double sized neighbors have to be treated differently, because their factor depends on the relative position of the cell. Basically all double sized neighbors are treated as same sized cells, thus with a prefactor of 1.0. However, a single larger cell neighboring over a face might be the same over edges or corners due to the relative position of the cell towards the double sized neighbor. As mentioned above P4EST does not return these neighbors. E.g. if P4EST does not return cells over the x-y-edge, there is either a double sized cell over the x-face or the y-face. This double sized cell has to be added to set $\{\mathbf{c}_n\}$ once more with the corresponding weight for the edge or corner. Applying this scheme to the example shown in Figure 5.8 gives the interpolation in Figure 5.9.

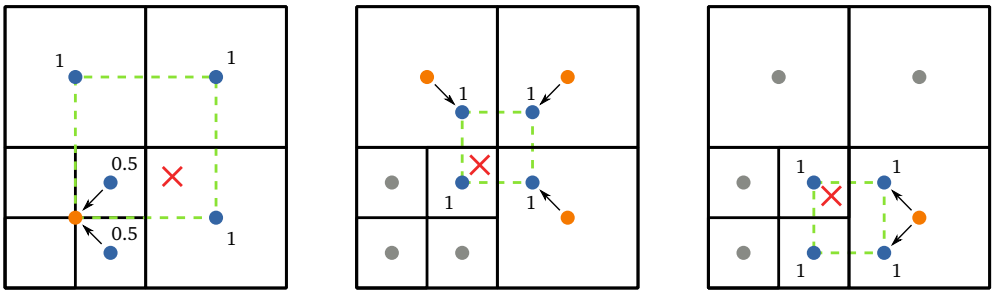


Figure 5.9: Interpolation with Equation (5.9) for the examples shown in Figure 5.8. On adaptive grids data points for differently sized neighbors are shifted (and averaged if necessary) to the position imposed by the linear interpolation. The position for interpolation is marked as red cross and the trilinear interpolation area is surrounded by a green line. The set $\mathbf{c}_m$ is given by the blue points. The respective scaling factors $\hat{\lambda}_m$ are printed alongside the points.

Overall, it is still a trilinear interpolation where cells with a differing size are averaged and the value's position is shifted by $\mathcal{O}(h)$. Taylor's theorem states for the general linear approximation that

$$f(x+\varepsilon) = f(x) + f'(x)\cdot\varepsilon + \mathcal{O}(\varepsilon^2)\ . \tag{5.11}$$

For two given values at the positions a and b with distance h (5.11) yields the one-dimensional linear interpolation

$$f(a+\lambda\cdot h) = (1-\lambda)\cdot f(a) + \lambda\cdot f(b) + \mathcal{O}(h^2)\ . \tag{5.12}$$

For adaptive refined neighbors $|a-b|$ is not equal to the mesh width h. Due to the 2:1 balancing it is either $\frac{3}{2}h$ or $\frac{3}{4}h$ for half or double sized neighbors, respectively. Assuming that $|a-b|$ is h by shifting the positions makes an error of $\mathcal{O}(h)$. This constant shifting of cell values from cell midpoints to the position required by the trilinear interpolation thus reduces the

accuracy order from $\mathcal{O}(h^2)$ to $\mathcal{O}(h)$. The order of the resulting interpolation scheme is equal to the constant interpolation in adaptively refined areas. In locally regular parts, however, it is still a first order interpolation.

For distributed adaptive grids it is possible to have the process domain boundary in regions of refinement such that cells in the ghost layer are not equally refined compared to the adjacent local cells. Coupling particles located in ghost cells raise additional effort as the adaptive neighborhood has to be computed. The first cell for interpolation is always given by the particle position. In the regular case it is sufficient to search for a direct match of the remaining seven global space-filling indices for the respective neighbor cells. Half sized cells with respect to the ghost cell are mapped to the same sized parent by using the level of the coarser ghost cell as reference for the global Morton index. Thus, all adjacent cells on a finer level are found without any change in the algorithm. Coarser cells, however, cover a larger area and it is not mandatory that their global index is equal to the computed neighbor index. The actual index of the double sized cell and the computed one are aligned using the local property of the space-filling index. Algorithm 5.6 sketches a compare operation to check if a quadrant overlaps with respect to a given global Morton index for a specific reference level. This algorithm is used to find all needed neighboring cells for a ghost cell considering the relative position of the particle.

Algorithm 5.6 Checks the overlap of a given P4EST quadrant with a global Morton index on a specific reference level.

```
1:  function QuadIndexOverlap(quad q, int idx, int ref_level)
2:      qidx ← GlobalQuadIdx(q, ref_level)
3:      range ← 1
4:      if q.level < ref_level then                ▷ Quadrant is coarser than reference level
5:          range ← 2^(3·(ref_level−q.level))      ▷ Number of indices covered by coarser quadrant
6:      end if
7:      if qidx ≤ idx < qidx + range then
8:          return 1
9:      end if
10:     return 0
11: end function
```

CHAPTER 6

Results

The described algorithms for the P4EST based molecular dynamic simulation shown in Chapter 4.3 are implemented in ESPRESSO as result of this work. The methods for coupling explained in Section 5, however, are not fully implemented and tested yet, due to the simultaneously ongoing development of the adaptive lattice-Boltzmann method in ESPRESSO. Algorithms in Section 5.2 for coupling MD and LB grids with arbitrary mesh width are a mere concept and are not tested and verified. The coupling is implemented for the adaptive lattice-Boltzmann method, but it is only tested on uniformly refined LB grids, because of a systematic error in the here used lattice-Boltzmann implementation at boundaries of refined areas that was discovered during this work. A further restriction of the implementation is a simplified version of Algorithm 5.5 for the computation of the finest common tree that assumes to have a finer LB grid in the whole domain compared to the MD grid sharing the same `p4est_connectivity`. The results presented in this chapter aim to cover representative simulation scenarios. The test cases are chosen such that the simulation results indicate potential errors in the code and validate important features and special cases of the algorithms.

6.1 Pure Molecular Dynamic Simulations

One of the fundamental algorithms for the particle coupling with adaptive grids is the molecular dynamic simulation with a P4EST based domain decomposition. The molecular dynamic code is the basic component of the ESPRESSO software. Hence, it is crucial to investigate the newly developed algorithms and their implementation with suitable test cases. The described algorithms do not alter the actual molecular dynamic time integration in ESPRESSO, but they redefine the memory management and the communication scheme for the simulation. Thus, a good test scenario is one of the classical models for statistical mechanics, the *NVE* or *microcanonical* ensemble, because errors in the neighborhood and ghost connectivity lead to a diverging energy. In the NVE ensemble the movement of a fixed number of particles (N) is simulated by using the Lennard-Jones potential in a given volume (V) such that the total energy (E) of the system is conserved. It can be seen as the natural ensemble for the linked cells algorithms, because both the number of particles and the simulation box are initial parameters. The particle set used by the script (see A.1) for this test series consists of 1000 particles and it is generated by minimizing the energy of the randomly placed particles and equilibrate them afterwards. As seen in Figure 6.1, the total energy of the system is conserved

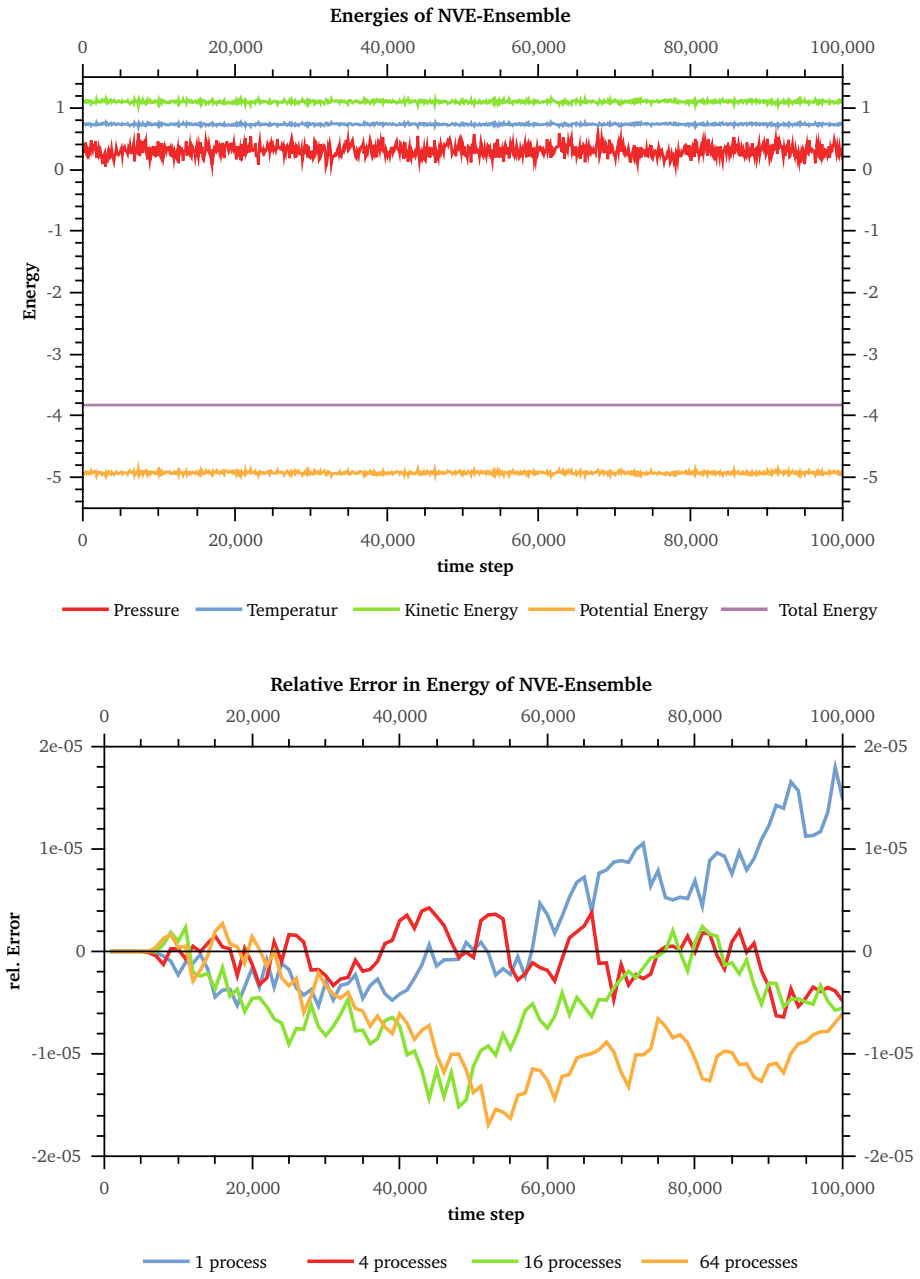


Figure 6.1: NVE ensemble with 1000 particles distributed with a mean density of 0.8442 and a time step of 0.001. Top: Averaged energies (temperature, kinetic, potential and total energy) and the pressure plotted over the simulation steps. The total energy is conserved despite a minimal fluctuation. The other quantities have a higher fluctuation around their mean. Bottom: Relative error of the total energy in the system plotted over integration steps with respect to the original ESPRESSO code. The results are simulated in parallel on 1, 4, 16, and 64 processes.

over 10^6 time integration steps as expected. The algorithm yields conforming results for the highly aggregated total energy in parallel simulations with up to 64 process with respect to the unchanged original ESPResSo implementation. The first 5000 integration steps for all runs are identical with respect to errors in order of the machine tolerance. Afterwards, the parallel original code as well as the p4est implementation produce results that fluctuate around the same mean value of -3.82 with standard deviations between $7.34 \cdot 10^{-6}$ and $1.34 \cdot 10^{-5}$ over all time samples. Thus, the relative error seen in Figure 6.1 lies within the fluctuation range of the total energy.

In a further test setting for the p4est based linked cells method the scaling properties of the developed algorithms are investigated. The NVE ensemble uses 1000 homogeneously distributed particles per process on the *Hazel Hen* super computer of the *HLRS* running 24 processes per node. Thus, the results represent the weak scaling of the algorithm with up to 512 nodes running 12288 processes. The runtime for both, the p4est based and the default ESPResSo, are plotted over the number of used nodes in Figure 6.2. The results shown are averaged over multiple node allocations and test runs. Overall, the developed decomposition scheme using p4est performs better than the original ESPResSo implementation, because of the improved methods for particle communication. The reason for the peak in the runtime for 128 nodes (3072 processes), however, is still uncertain and needs further research.

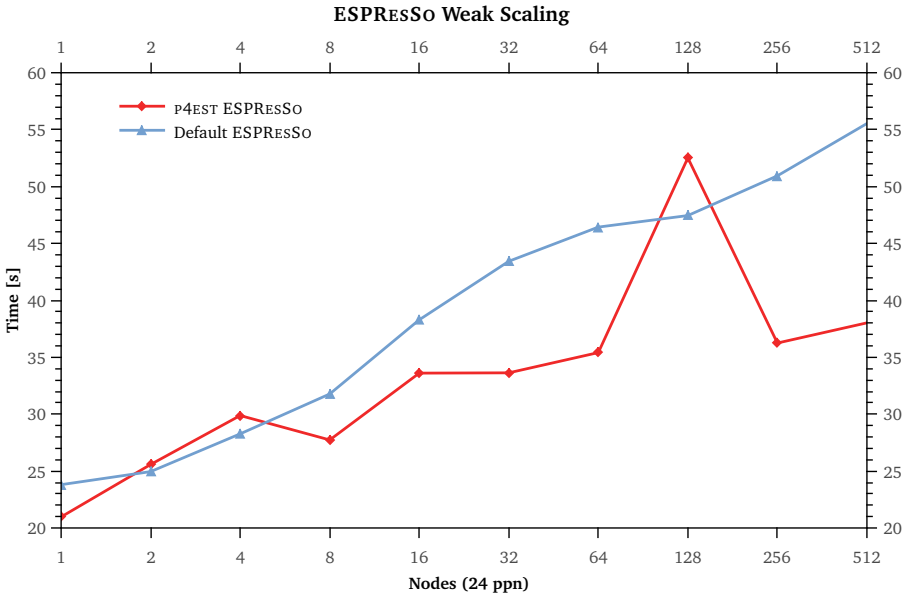


Figure 6.2: Weak scaling for both the p4est based (red line with square marks) and the original ESPResSo (blue line with triangle marks) implementation. The runtime of the homogeneous NVE ensemble with 1000 particles per process is plotted over the number of used compute nodes running 24 processes each.

6.2 Particle-Fluid Coupling Simulations

The developed methods for particle coupling together with the included velocity interpolation and extrapolation scheme are tested with simplistic scenarios by comparing the simulation result of the P4EST based method with the original ESPRESSO implementation. To separate potential error sources, only a single particle is coupled with the fluid. The setting seen in Figure 6.3 using the script A.2 simulates a particle free of external forces that is dragged by a flow in a channel. The fluid is accelerated by an external force. The force is coupled to the particle by the fluid's viscosity. Thus, one expects that the resulting total force on the particle is smaller than the external force on the fluid and the flow near the particle is slowed down due to the slower particle movement. In Table 6.1 the simulation results for this scenario are shown with mesh widths of 0.5, 0.25, and 0.125 on a uniformly refined lattice-Boltzmann grid. The grid used by the linked cells algorithm has a constant mesh size of 0.5 for all test runs. As seen, the results of the newly developed algorithms perfectly match the reference solution of the original ESPRESSO implementation up to numerical errors in order of the machine error for double precision floating point values.

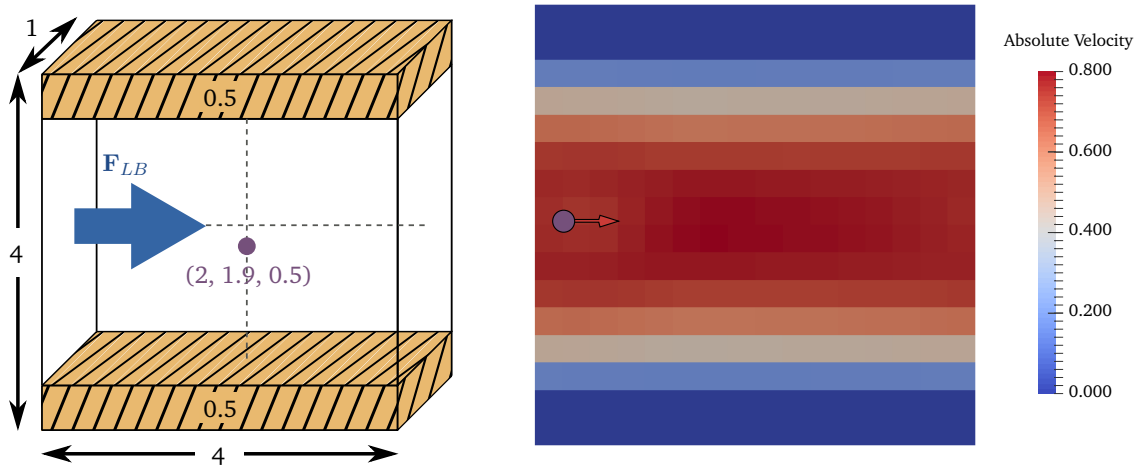


Figure 6.3: Left: Setting for particle coupling with a force driven fluid (see A.2) in a channel. The particle is placed near the box center and is transported by the fluid. The fluid is driven by an external force of 0.01 and has a viscosity of 0.005. Right: Cross section of the fluid at the final simulation time $T = 100$ with a mesh width of 0.25. The particle has not fully reached the fluid velocity. Thus, the fluid in this area is slowed down.

Test			Position			Fluid Velocity			Particle Velocity		
h	Δt		x	y	z	x	y	z	x	y	z
$\frac{1}{2}$	$\frac{1}{100}$	Default	41.77	1.875	.500	.788	$-.619\,\text{E}^{-3}$	0.000	.775	$-.604\,\text{E}^{-3}$	0.000
		P4EST	41.77	1.875	.500	.789	$-.619\,\text{E}^{-3}$	0.000	.775	$-.604\,\text{E}^{-3}$	0.000
		L_2-Error	0.00000			$.26748\,\text{E}^{-15}$			$.83802\,\text{E}^{-16}$		
$\frac{1}{4}$	$\frac{1}{200}$	Default	36.63	2.036	.500	.750	$-.361\,\text{E}^{-3}$	$-.101\,\text{E}^{-15}$	.736	$-.172\,\text{E}^{-3}$	$-.110\,\text{E}^{-15}$
		P4EST	36.63	2.036	.500	.750	$-.361\,\text{E}^{-3}$	$-.572\,\text{E}^{-16}$	.736	$-.172\,\text{E}^{-3}$	$-.385\,\text{E}^{-16}$
		L_2-Error	0.00000			$.59897\,\text{E}^{-16}$			$.99553\,\text{E}^{-16}$		
$\frac{1}{8}$	$\frac{1}{400}$	Default	27.80	1.870	.500	.615	$-.783\,\text{E}^{-3}$	$-.262\,\text{E}^{-16}$	.605	$-.796\,\text{E}^{-3}$	$-.336\,\text{E}^{-16}$
		P4EST	27.804	1.870	.500	.615	$-.783\,\text{E}^{-3}$	$.878\,\text{E}^{-16}$	.605	$-.796\,\text{E}^{-3}$	$.803\,\text{E}^{-16}$
		L_2-Error	0.00000			$.31430\,\text{E}^{-15}$			$.24269\,\text{E}^{-15}$		

Table 6.1: Simulation results for the setting seen in Figure 6.3 for three different LB mesh widths. The fluid is evaluated at the particle's final position. For the results of the P4EST based algorithms the relative L_2 error is given with respect to the original ESPRESSO implementation.

The simulation is repeated for one and up to 64 processes and on different maximal refinement levels, in order to validate the functionality of the interpolation and extrapolation algorithms in parallel and not fully refined grids. Although the LB mesh width for these tests is constant, the maximal refinement level influences the results, because it changes the time step and reference values of the algorithms. Due to the subcycling scheme of the adaptive lattice-Boltzmann method, the streaming step on coarser levels is performed later compared to the non-adaptive scheme. This has numerical effects on the solution seen in Figure 6.4, as the adaptive coupling accesses flow velocities that are older by $\mathcal{O}(\Delta t)$ on not fully refined parts of the grid. The error plotted over the number of used processes shows that the method works with an accuracy in the order of the machine tolerance up to 16 processes. The reason for the drastic increase in the relative error for 32 and 64 processes, however, needs further investigation. Pure lattice-Boltzmann simulations with the same fluid properties but without any particles have shown unexpected behavior and system runtime errors for parallel settings with more than 16 processes as well. Thus, it is yet unclear whether the error lies in the coupling method, in the lattice-Boltzmann algorithm, or in a bad choice of scenario parameters.

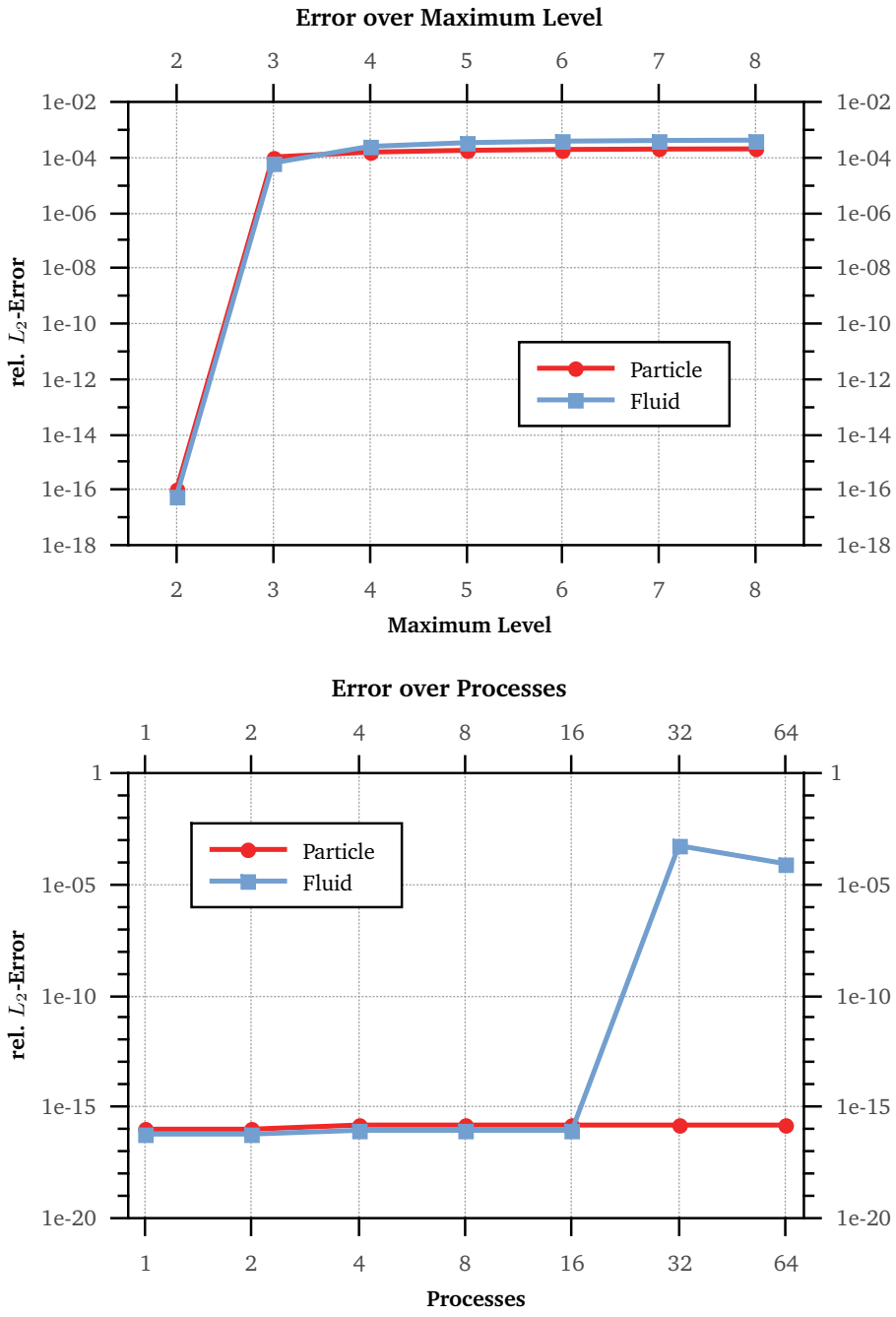


Figure 6.4: Relative L_2 error of the particle and flow velocities at the final time $T = 100$ for the simulation setting given in Figure 6.3 (see A.2) comparing the P4EST and the original algorithms. The used mesh widths are 0.25 for the lattice-Boltzmann grid and 0.5 for the MD cells in all test runs. The flow velocity is evaluated at the particle position. Top: Relative error for a variation of the maximal possible refinement level on a single process. Bottom: Relative error over the number of used parallel processes with maximal refinement on level two.

The second testing scenario visualized in Figure 6.5 (see Script A.3) is similar to the setting seen above. But instead of applying the force to the fluid the particle is driven by an external force. In contrast to the scenario with the force driven fluid, one expects the particle to be faster than the fluid, because it passes the force to the fluid by the viscous coupling. The simulation results of the particle velocity and the interpolated flow velocity at the particle's final position for regular refined LB grids with mesh widths of 0.5, 0.25, and 0.125 in Table 6.2 perfectly match the data generated with the original ESPRESSO implementation up to numerical errors in the order of the machine tolerance.

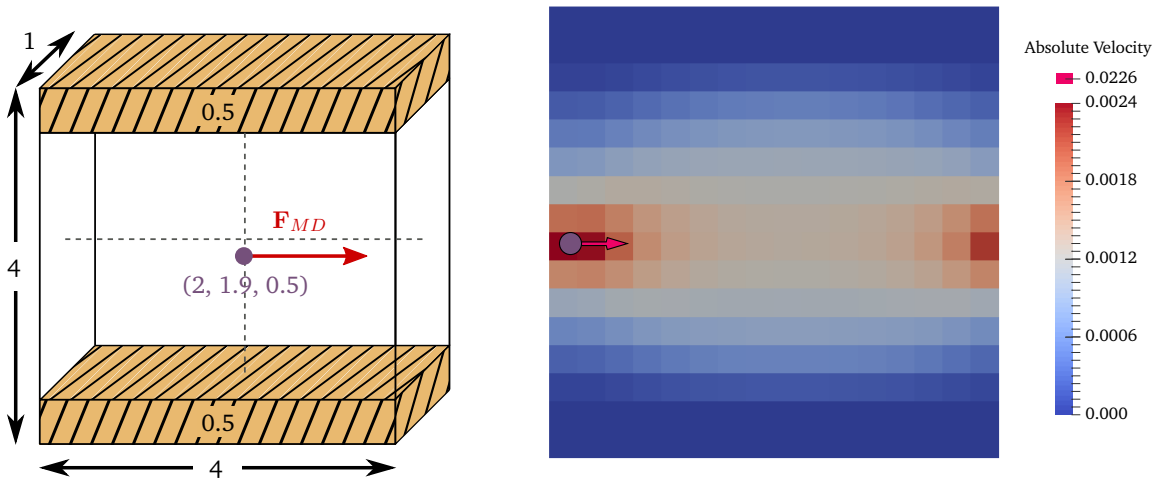


Figure 6.5: Left: Setting for fluid coupling with a force driven particle (see A.3) in a channel. The particle is placed near the box center and accelerates the fluid with a viscosity of 1. The particle is driven by an external force of 0.01. Right: Cross section of the fluid at the final simulation time $T = 100$ with a mesh width of 0.25. The particle moves faster than the fluid at the beginning and it accelerates the fluid.

As above, the simulation is repeated for different maximal refinement levels and parallel processes. Both test series are performed with a base mesh width for the lattice-Boltzmann method of 0.25. The original code and the here developed P4EST based implementation are compared by computing the L_2 error shown in Figure 6.6 of the particle and flow velocity after the last integration step. The fluid is evaluated at the particle's final position. Variations of the maximal refinement level show the same behavior as for the previous test case. For a maximal level greater than the grid's base level, thus finer MD integration steps compared to the coarse LB steps, the relative error increases in a similar way as for the test scenario with the force driven fluid above. One possible reason for this increasing error is the time-displaced streaming step by $\mathcal{O}(\Delta t)$ of the coarse lattice-Boltzmann cells compared to the non-adaptive algorithm. In contrast to the results shown above, the accuracy for all parallel simulations up to 64 processes stay in the order of the machine tolerance.

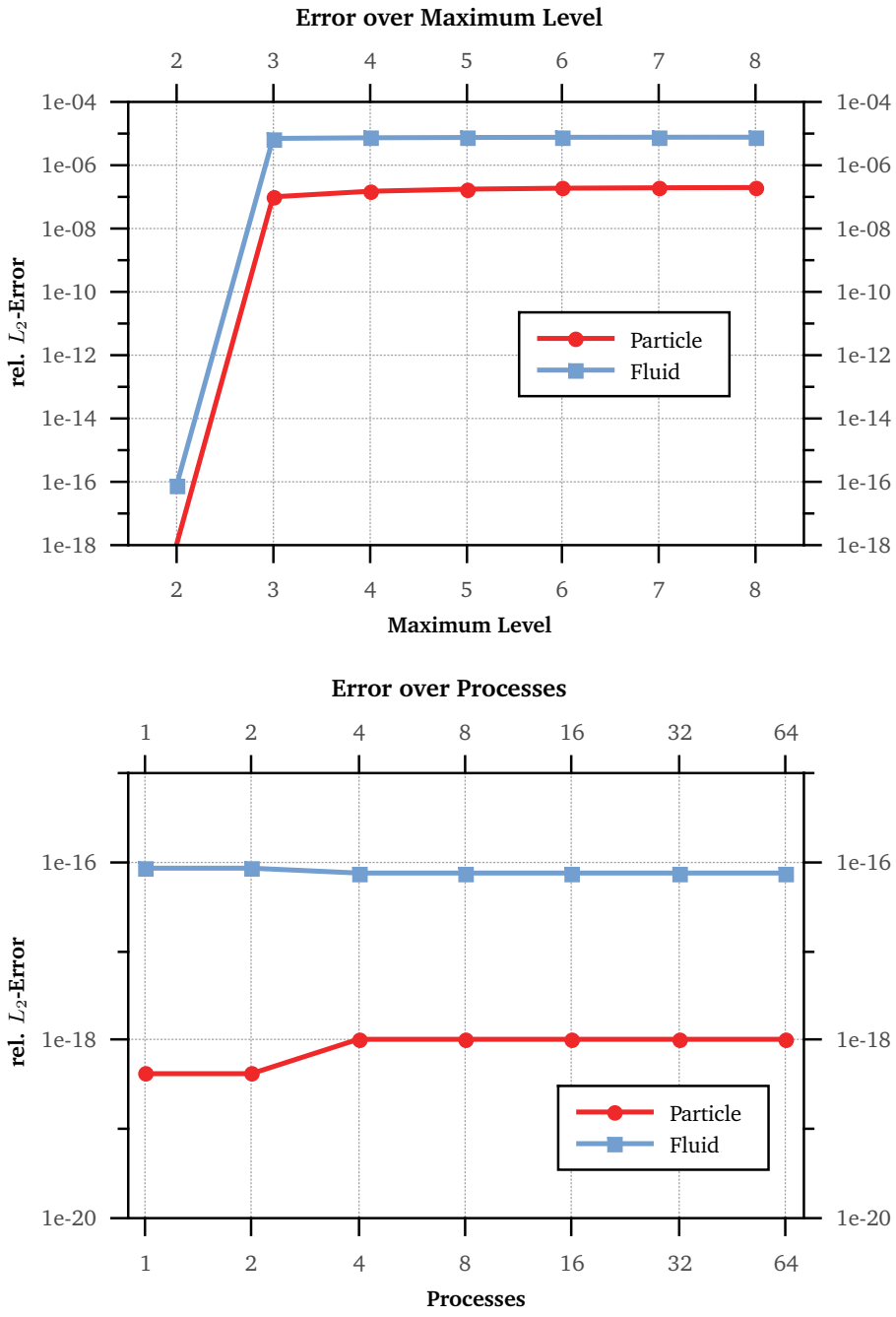


Figure 6.6: Relative L_2 error of the particle and flow velocities at the final time $T = 100$ for the simulation setting given in Figure 6.5 (see A.3) comparing the P4EST and the original algorithms. The used mesh widths are 0.25 for the lattice-Boltzmann grid and 0.5 for the MD cells in all test runs. The flow velocity is evaluated at the particle position. Top: Relative error for a variation of the maximal possible refinement level on a single process. Bottom: Relative error over the number of used parallel processes with maximal refinement on level two.

Test			Position			Fluid Velocity			Particle Velocity		
h	Δt		x	y	z	x	y	z	x	y	z
$\frac{1}{2}$	$\frac{1}{100}$	Default	4.111	1.900	.500	$.159\,\text{E}^{-2}$	$.248\,\text{E}^{-6}$	0.000	$.216\,\text{E}^{-1}$	$.258\,\text{E}^{-6}$	0.000
		P4EST	4.111	1.900	.500	$.157\,\text{E}^{-2}$	$.248\,\text{E}^{-6}$	0.000	$.216\,\text{E}^{-1}$	$.258\,\text{E}^{-6}$	0.000
		L_2-Error		0.00000			$.36238\,\text{E}^{-16}$			$.15942\,\text{E}^{-18}$	
$\frac{1}{4}$	$\frac{1}{200}$	Default	4.208	1.900	.500	$.260\,\text{E}^{-2}$	$.307\,\text{E}^{-6}$	$-.379\,\text{E}^{-19}$	$.226\,\text{E}^{-1}$	$.311\,\text{E}^{-6}$	$-.126\,\text{E}^{-19}$
		P4EST	4.208	1.900	.500	$.260\,\text{E}^{-2}$	$.307\,\text{E}^{-6}$	$-.122\,\text{E}^{-18}$	$.226\,\text{E}^{-1}$	$.311\,\text{E}^{-6}$	$-.223\,\text{E}^{-19}$
		L_2-Error		0.00000			$.75435\,\text{E}^{-16}$			$.10254\,\text{E}^{-17}$	
$\frac{1}{8}$	$\frac{1}{400}$	Default	4.350	1.900	.500	$.413\,\text{E}^{-2}$	$.352\,\text{E}^{-6}$	$-.610\,\text{E}^{-19}$	$.241\,\text{E}^{-1}$	$.359\,\text{E}^{-6}$	$-.242\,\text{E}^{-20}$
		P4EST	4.350	1.900	.500	$.413\,\text{E}^{-2}$	$.352\,\text{E}^{-6}$	$-.542\,\text{E}^{-19}$	$.241\,\text{E}^{-1}$	$.359\,\text{E}^{-6}$	$.115\,\text{E}^{-21}$
		L_2-Error		0.00000			$.17272\,\text{E}^{-17}$			$.26172\,\text{E}^{-18}$	

Table 6.2: Simulation results for the setting seen in Figure 6.5 for three different LB mesh widths. The fluid is evaluated at the particle's final position. For the results of the P4EST based algorithms the relative L_2 error is given with respect to the original ESPRESSO implementation.

Furthermore, a standard setting for particle-fluid coupling is illustrated in Figure 6.7 and described by the script A.4. The scenario consists of a single particle with fixed position and velocity in the center of the simulation box. The fully periodic fluid is accelerated by the viscous coupling with the particle.

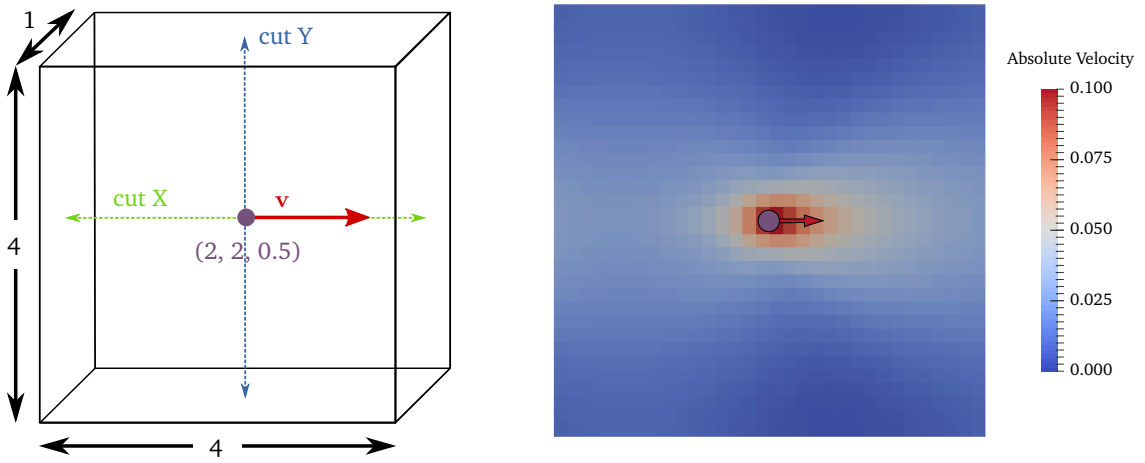


Figure 6.7: Left: Setting for the viscous particle-fluid coupling of a particle with fixed position and velocity (see A.4). The particle is located in the center of the fluid box. The cross section paths for the error analysis are marked with *cut X* (green) and *cut Y* (blue). Right: Cross section of the fluid with a viscosity of 0.01 and a LB mesh width of 0.125 at the final simulation time $T = 100$.

The simulation is performed on four parallel process with a regular refined LB grid on level three, thus a mesh width of 0.125. Due to the domain decomposition of both, MD and LB grid, the particle appears on all four processes either in local or ghost cells. The interpolated flow velocity after the last integration step and its relative errors between the original ESPRESSO code and the adaptive P4EST implementation is plotted over the cross section paths *cut X* and *cut Y* shown in Figure 6.8. As can be seen, both methods yield the same results up to numerical fluctuations in the order of machine epsilon for double precision floating point values.

The results shown in this chapter are quite promising as almost all perfectly match the competing non-adaptive implementation for settings with identical parameters and with maximal refined LB grids. However, the discrepancy for not fully refined regular grids obviously needs further investigation, although one known reason for it is the time-displacement of the streaming step as explained above. Furthermore, no tests of the developed coupling algorithms with truly adaptive grids are performed so far. Earlier test series with the same settings described in this section are not shown. Nevertheless, they are crucial for the development, since errors in the implementations as for example missing ghost communications of the lattice-Boltzmann data directly before computing viscous coupling forces are discovered via this tests.

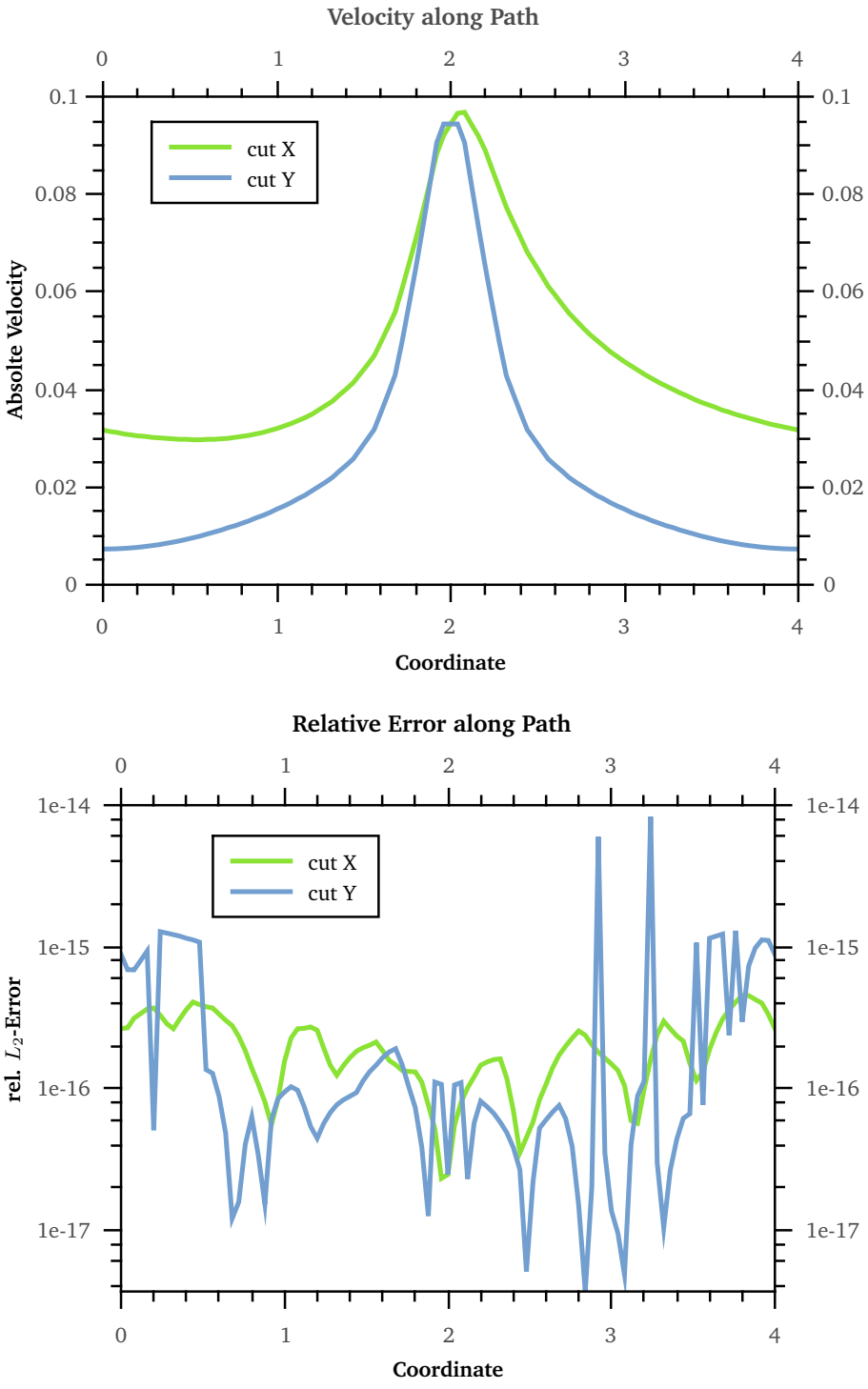


Figure 6.8: Results of the test scenario visualized in Figure 6.7 (see A.4). Top: the interpolated flow velocity along the cross section paths *cut X* and *cut Y* after the final integration step at $T = 100$. Bottom: Relative L_2 error of the velocity along the paths between the original ESPRESSO implementation and the adaptive P4EST method.

CHAPTER 7

Discussion and Future Work

In this work a concept for coupling molecular dynamic simulations with the spatial adaptive lattice-Boltzmann fluid simulation is presented. The developed algorithms are designed to work in parallel and make minimal changes to the existing code of the ESPRESSO simulation software. Particle coupling involves two different types of grids, the regular linked cells grid for the molecular dynamic simulation with short-ranged interaction potentials and the adaptive grid of the fluid simulation.

The adaptive lattice-Boltzmann method discussed in Chapter 3.2 is based on the P4EST software library for tree-structured grids. Due to the space-filling Morton order imposed by P4EST, the resulting domain decomposition scheme for parallel computations has highly non-regular shaped subdomains. The approach explained in Chapter 4.3 aims to realize the same P4EST based domain decomposition for the linked cells algorithm, in order to minimize additional interprocess communication of particle and fluid data caused by diverging domain decompositions for both grids. The developed algorithms for interprocess particle migration, ghost layer updates, and the decomposition based on the Morton curve make an asynchronous communication scheme mandatory. The results for the runtime of pure molecular dynamic simulations presented in Chapter 6.1 show that the developed methods perform even better for weak scaling than the competing reference implementation of the original ESPRESSO code. Thanks to the highly flexible software design of the ESPRESSO software, only directly affected components, i.e. the domain decomposition and the ghost communication, needed to be changed, while the whole particle management and the time integration methods are untouched. However, not all explained features for the P4EST based linked cells algorithms are implemented due to the limited time for this work. The ordered structure of the grids along the space-filling curve allows efficient access in $\mathcal{O}(\log N)$ by using binary search or even in $\mathcal{O}(1)$ for parts of regular grids. With respect to simplicity reasons and to reduce possible error sources, these efficient methods are not used and instead a basic linear search that is in $\mathcal{O}(N)$ is implemented.

Furthermore, two approaches to distribute lattice-Boltzmann and linked cells grids with differing mesh widths and topologies are described in Chapter 5.1 and 5.2. The method based on the finest common tree is restricted to grids that share the same topology, i.e. that are based on the same `p4est_connectivity`, where the other approach is capable of handling grids with fully arbitrary mesh widths. The latter method, however, requires additional preprocessing steps and a more sophisticated particle exchange routine. Only a simplified version of the

algorithm computing the finest common tree is implemented in the ESPRESSO simulation software, as it covers all cases needed for the development of the particle-fluid coupling on adaptive LB grids. The algorithms need to be extended, in order to cover a wider range of simulations for actual application scenarios.

A major issue for the particle coupling on adaptive refined grids is the interpolation of the fluid velocity at the particle's position and the later extrapolation of the viscous force on the involved fluid cells. On fully regular grids and in locally regular parts of adaptive grids the trilinear interpolation scheme can be applied. However, in refined areas of adaptive grids the extended interpolation method explained in Chapter 5.3 is used. This method is conforming with the trilinear scheme in regular areas, but it is one accuracy order lower in refined areas.

The coupling is only tested on regular refined grids, because the adaptive lattice-Boltzmann implementation in ESPRESSO is still under development and the version used here has issues in refined areas. The results for fully refined grids seen in Chapter 6.2 are promising as they are equal the reference solutions generated by the original ESPRESSO implementation up to errors in order of the machine tolerance. Scenarios with not fully refined grids are a first step towards adaptivity, because these settings resemble coarse areas of locally refined grids. However, their solutions diverges from the corresponding reference simulation. The algorithmic difference of the adaptive and the regular lattice-Boltzmann method is a possible explanation for this error. Thus, although the developed method is capable of coupling MD simulations with adaptive LB grids, further investigation is needed for fully adaptive scenarios. A further restriction of the current implementation is the maximal LB mesh width of 1 prescribed by the P4EST topology, because it automatically forces the MD cells to be smaller than 1 as well. However, the MD mesh is prescribed by the cut-off radius of the interactive potential for which this is a severe constraint.

Appendix A

Scripts

Listing A.1: Script for NVE simulation of a pre-equilibrated set of 1000 particles with a mean density of 0.8442

```
cellsystem domain_decomposition -no_verlet_list
setmd skin 0.1

set n_part 1000
set density 0.8442
set boxl [expr pow($n_part/$density,1.0/3.0)+2*0.1]
setmd box_l $boxl $boxl $boxl

inter 0 0 lennard-jones 1.0 1.0 2.5 auto 0.0
thermostat off

set in [open "particle.dat" "r"]
blockfile $in read auto
close $in

setmd time_step 0.001

for { set i 0 } { $i < 1000 } { incr i } {
  integrate 100
  set energies [analyze energy]
  set total_en [expr [lindex $energies 0 1]/$n_part]
  set kinetic [expr [lindex $energies 1 1]/$n_part]
  set potential [expr [lindex $energies 2 3]/$n_part]
  set pres [analyze pressure total]
  set temperature [expr [lindex $energies 1 1]/(1.5*[setmd n_part])]
  puts "$pres $temperature $kinetic $potential $total_en"
}

exit
```

Listing A.2: Script for coupling with force driven fluid

```
cellsystem domain_decomposition -no_verlet_list
setmd skin 0.1
inter 0 0 lennard-jones 1.0 0.1 0.4 auto 0.0
setmd box_l 4 4 1

part 0 pos 2.0 1.9 0.5 q 0.0 type 0 v 0 0 0 ext_force 0.0 0.0 0.0

set use_adapt_lbm 1

set base_level 2
set max_level 2
set level_factor [expr pow(2, $max_level - 1)]

set dt 0.01

thermostat lb 0
setmd time_step [expr $dt/$level_factor]

if {$use_adapt_lbm > 0} {
  lbadapt-init $base_level
  lbadapt_set_max_level $max_level
  lbfluid cpu agrid [expr 1./pow(2, $max_level)] dens 1.0 visc 0.005 tau [expr
      $dt/$level_factor] friction 0.5 ext_force 0.01 0.0 0.0
  lbadapt-reset-fluid
} else {
  lbfluid cpu agrid [expr 1./pow(2, $base_level)] dens 1.0 visc 0.005 tau [expr
      $dt/pow(2, $base_level - 1)] friction 0.5 ext_force 0.01 0.0 0.0
}
lbboundary wall dist 0.5 normal 0 1 0
lbboundary wall dist -3.5 normal 0 -1 0

integrate [expr 10000 * int($level_factor)]

set px [lindex [part 0 print pos] 0]
set py [lindex [part 0 print pos] 1]
set pz [lindex [part 0 print pos] 2]
set $fluid_velocity [lbfluid print_interpolated_velocity $px $py $pz]
puts "$px $py $pz $fluid_velocity [part 0 print v]"

exit
```

Listing A.3: Script for coupling with force driven particle

```
cellsystem domain_decomposition -no_verlet_list
setmd skin 0.1
inter 0 0 lennard-jones 1.0 0.1 0.4 auto 0.0
setmd box_l 4 4 1

part 0 pos 2.0 1.9 0.5 q 0.0 type 0 v 0 0 0 ext_force 0.01 0.0 0.0

set use_adapt_lbm 1

set base_level 2
set max_level 2
set level_factor [expr pow(2, $max_level - 1)]

set dt 0.01

thermostat lb 0
setmd time_step [expr $dt/$level_factor]

if {$use_adapt_lbm > 0} {
  lbadapt-init $base_level
  lbadapt_set_max_level $max_level
  lbfluid cpu agrid [expr 1./pow(2, $max_level)] dens 1.0 visc 1.0 tau [expr
      $dt/$level_factor] friction 0.5 ext_force 0.0 0.0 0.0
  lbadapt-reset-fluid
} else {
  lbfluid cpu agrid [expr 1./pow(2, $base_level)] dens 1.0 visc 1.0 tau [expr
      $dt/pow(2, $base_level - 1)] friction 0.5 ext_force 0.0 0.0 0.0
}
lbboundary wall dist 0.5 normal 0 1 0
lbboundary wall dist -3.5 normal 0 -1 0

integrate [expr 10000 * int($level_factor)]

set px [lindex [part 0 print pos] 0]
set py [lindex [part 0 print pos] 1]
set pz [lindex [part 0 print pos] 2]
set $fluid_velocity [lbfluid print_interpolated_velocity $px $py $pz]
puts "$px $py $pz $fluid_velocity [part 0 print v]"

exit
```

Listing A.4: Script for coupling with fixed particle

```
cellsystem domain_decomposition -no_verlet_list
setmd skin 0.1
inter 0 0 lennard-jones 1.0 0.1 0.4 auto 0.0
setmd box_l 4 4 1

part 0 pos 2.0 2.0 0.5 q 0.0 type 0 v 0.1 0 0 ext_force 0.0 0.0 0.0 fix 1 1 1

set use_adapt_lbm 1

set base_level 3
set max_level 3
set level_factor [expr pow(2, $max_level - 1)]

set dt 0.01

thermostat lb 0
setmd time_step [expr $dt/$level_factor]

if {$use_adapt_lbm > 0} {
  lbadapt-init $base_level
  lbadapt_set_max_level $max_level
  lbfluid cpu agrid [expr 1./pow(2, $max_level)] dens 1.0 visc 0.01 tau [expr
      $dt/$level_factor] friction 0.5 ext_force 0.01 0.0 0.0
  lbadapt-reset-fluid
} else {
  lbfluid cpu agrid [expr 1./pow(2, $base_level)] dens 1.0 visc 0.01 tau [expr
      $dt/pow(2, $base_level - 1)] friction 0.5 ext_force 0.01 0.0 0.0
}
lbboundary wall dist 0.5 normal 0 1 0
lbboundary wall dist -3.5 normal 0 -1 0

integrate [expr 10000 * int($level_factor)]

for {set i 0} {$i <= 200} {incr i} {
  set $cut_X [lbfluid print_interpolated_velocity [expr $i*0.02] 2.0 0.5]
  set $cut_Y [lbfluid print_interpolated_velocity 2.0 [expr $i*0.02] 0.5]
  puts "$i $cut_X $cut_Y"
}

exit
```

APPENDIX

Bibliography

[AD69] A. Z. Akcasu, J. J. Duderstadt. Derivation of kinetic equations from the generalized Langevin equation. *Physical Review*, 188(1):479, 1969. (Cited on page 17)

[AD99] 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):8225–8239, 1999. (Cited on pages 10, 43 and 46)

[ALK+ed] A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Roehm, P. Kosovan, C. Holm. ESPResSo 3.1 – Molecular Dynamics Software for Coarse-Grained Models. In M. Griebel, C. Rieger, M. A. Schweitzer, editors, *Proceedings of the Sixth International Workshop on Meshfree Methods for Partial Differential Equations*, Lecture Notes in Computational Science and Engineering. Springer, Berlin, Germany, submitted. (Cited on pages 9, 11 and 25)

[AT89] M. Allen, D. Tildesley. *Computer Simulation of Liquids*. Oxford Science Publ. Clarendon Press, 1989. (Cited on page 28)

[AW59] B. J. Alder, T. E. Wainwright. Studies in molecular dynamics. I. General method. *The Journal of Chemical Physics*, 31(2):459–466, 1959. (Cited on page 25)

[BCX+06] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, et al. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 84. ACM, 2006. (Cited on page 25)

[BGK54] 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):511, 1954. (Cited on page 19)

[BGL91] C. Bardos, F. Golse, D. Levermore. Fluid dynamic limits of kinetic equations. I. Formal derivations. *Journal of statistical physics*, 63(1):323–344, 1991. (Cited on page 17)

[BL04] C. K. Birdsall, A. B. Langdon. *Plasma physics via computer simulation*. CRC Press, 2004. (Cited on page 26)

[Buc10] M. Buchholz. *Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen*. Verlag Dr. Hut, 2010. (Cited on pages 10 and 30)

[BWG11] 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):1103–1133, 2011. doi:10.1137/100791634. URL http://dx.doi.org/10.1137/100791634. (Cited on pages 9 and 12)

[Cer88] C. Cercignani. The Boltzmann equation. In *The Boltzmann Equation and Its Applications*, pp. 40–103. Springer, 1988. (Cited on page 18)

[CFL28] R. Courant, K. Friedrichs, H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische annalen*, 100(1):32–74, 1928. (Cited on page 40)

[CK10] M. Connor, P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *Visualization and Computer Graphics, IEEE Transactions on*, 16(4):599–608, 2010. (Cited on page 28)

[CKW+13] L. Cheng, R. Kirsch, A. Wiegmann, P.-C. Gervais, N. Bardin-Monnier, D. Thomas. PleatLab: A pleat scale simulation environment for filtration simulation. In *Proceedings of the FILTECH 2013 Conference*. 2013. (Cited on page 9)

[CP07] C. Chipot, A. Pohorille. Calculating free energy differences using perturbation theory. In *Free energy calculations*, pp. 33–75. Springer, 2007. (Cited on page 25)

[Deg04] P. Degond. Macroscopic limits of the Boltzmann equation: a review. In *Modeling and Computational Methods for Kinetic Equations*, pp. 3–57. Springer, 2004. (Cited on pages 17 and 18)

[DK93] B. Dünweg, K. Kremer. Molecular dynamics simulation of a polymer chain in solution. *The Journal of chemical physics*, 99(9):6983–6997, 1993. (Cited on page 43)

[DL09] B. Dünweg, A. J. Ladd. Lattice Boltzmann simulations of soft matter systems. In *Advanced Computer Simulation Approaches for Soft Matter Sciences III*, pp. 89–166. Springer, 2009. (Cited on pages 11 and 17)

[FPU55] E. Fermi, J. Pasta, S. Ulam. Studies of nonlinear problems. *Los Alamos Report LA-1940*, 978:977–988, 1955. (Cited on page 25)

[FS02] D. Frenkel, B. Smit. *Understanding molecular simulation : from algorithms to applications*. Academic Press, San Diego, 2nd edition, 2002. (Cited on pages 25, 27 and 31)

[GKZ07] M. Griebel, S. Knapek, G. Zumbusch. *Numerical Simulation in Molecular Dynamics*. Springer Science & Business Media, 2007. (Cited on pages 27 and 28)

[HEHB15] A. Heinecke, W. Eckhardt, M. Horsch, H.-J. Bungartz. *Supercomputing for Molecular Dynamics Simulations: Handling Multi-Trillion Particles in Nanofluidics*. Springer Publishing Company, Incorporated, 2015. (Cited on pages 25 and 31)

[HGE74] R. Hockney, S. Goel, J. Eastwood. Quiet high-resolution computer models of a plasma. *Journal of Computational Physics*, 14(2):148–158, 1974. (Cited on page 28)

[HKVDSL08] B. Hess, C. Kutzner, D. Van Der Spoel, E. Lindahl. GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3):435–447, 2008. (Cited on page 25)

[HRK+13] J. Höpfner, T. Richter, P. Košovan, C. Holm, M. Wilhelm. Seawater desalination via hydrogels: Practical realisation and first coarse grained simulations. In *Intelligent Hydrogels*, pp. 247–263. Springer, 2013. (Cited on page 11)

[IAKW14] G. Inci, A. Arnold, A. Kronenburg, R. Weeber. Modeling nanoparticle agglomeration using local interactions. *Aerosol Science and Technology*, 48(8):842–852, 2014. (Cited on page 11)

[IBWG15] T. Isaac, C. Burstedde, L. C. Wilcox, O. Ghattas. Recursive algorithms for distributed forests of octrees. *SIAM Journal on Scientific Computing*, 37(5):C497–C531, 2015. (Cited on page 12)

[Ise86] A. Iserles. Generalized leapfrog methods. *IMA Journal of Numerical Analysis*, 6(4):381–392, 1986. (Cited on page 26)

[Jon24] J. E. Jones. On the determination of molecular fields. II. From the equation of state of a gas. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 106, pp. 463–477. The Royal Society, 1924. (Cited on page 27)

[JR85] W. L. Jorgensen, C. Ravimohan. Monte Carlo simulation of differences in free energies of hydration. *The Journal of chemical physics*, 83(6):3050–3054, 1985. (Cited on page 25)

[KA15] M. Kuron, A. Arnold. Role of geometrical shape in like-charge attraction of DNA. *The European Physical Journal E*, 38(3):1–6, 2015. (Cited on page 11)

[KG90] K. Kremer, G. S. Grest. Dynamics of entangled linear polymer melts: A molecular-dynamics simulation. *The Journal of Chemical Physics*, 92(8):5057–5086, 1990. (Cited on page 43)

[KK05] M. Karplus, J. Kuriyan. Molecular dynamics and protein function. *Proceedings of the National Academy of Sciences of the United States of America*, 102(19):6679–6685, 2005. (Cited on page 25)

[LAMH06] H.-J. Limbach, A. Arnold, B. A. Mann, C. Holm. ESPResSo – An Extensible Simulation Package for Research on Soft Matter Systems. *Comput. Phys. Commun.*, 174(9):704–727, 2006. doi:10.1016/j.cpc.2005.10.005. (Cited on pages 9, 11 and 25)

[LBH+16] 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. In *Proceedings of the ECCOMAS Congress*, pp. 1–25. 2016. (Cited on pages 9, 11 and 22)

[LWKH16] M. J. Lehmann, J. Weber, A. Kilian, M. Heim. Microstructure Simulation as Part of Fibrous Filter Media Development Processes - From Real to Virtual Media. *Chemical Engineering & Technology*, 39(3):403–408, 2016. doi:10.1002/ceat.201500341. URL http://dx.doi.org/10.1002/ceat.201500341. (Cited on page 9)

[MKM+15] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, F. X. Giraldo. Strong scaling for numerical weather prediction at petascale with the atmospheric model numa. *arXiv preprint arXiv:1511.01561*, pp. 1–33, 2015. (Cited on page 12)

[Mor66] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966. (Cited on page 12)

[NBB+14] C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C. W. Glass, H. Hasse, et al. ls1 mardyn: The massively parallel molecular dynamics code for large systems. *Journal of chemical theory and computation*, 10(10):4455–4464, 2014. (Cited on page 25)

[PBW+05] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, K. Schulten. Scalable molecular dynamics with NAMD. *Journal of computational chemistry*, 26(16):1781–1802, 2005. (Cited on page 25)

[Pli95] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995. (Cited on page 25)

[PR92] C. Pierleoni, J.-P. Ryckaert. Molecular dynamics investigation of dynamic scaling for dilute polymer solutions in good solvent conditions. *The Journal of chemical physics*, 96(11):8539–8551, 1992. (Cited on page 43)

[Pro16] Prometeus GmbH. TOP500 Supercomputer Lists, 2016. URL https://www.top500.org/lists/2016/11/. (Cited on pages 26 and 31)

[RA12] D. Röhm, A. Arnold. Lattice boltzmann simulations on gpus with espresso. *The European Physical Journal-Special Topics*, 210(1):89–100, 2012. (Cited on page 11)

[Raa04] D. Raabe. Overview of the lattice Boltzmann method for nano-and microscale fluid dynamics in materials science and engineering. *Modelling and Simulation in Materials Science and Engineering*, 12(6):R13, 2004. (Cited on page 17)

[RDL77] P. Résibois, M. De Leener. *Classical kinetic theory of fluids*. Wiley, 1977. (Cited on page 17)

[RG65] S. A. Rice, P. Gray. *The statistical mechanics of simple liquids*. Interscience Publ., 1965. (Cited on page 17)

[RIH+07] B. J. Reynwar, G. Illya, V. A. Harmandaris, M. M. Müller, K. Kremer, M. Deserno. Aggregation and vesiculation of membrane proteins by curvature-mediated interactions. *Nature*, 447(7143):461–464, 2007. (Cited on page 11)

[RKDA06] M. Rohde, D. Kandhai, J. Derksen, H. Van den Akker. A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes. *International journal for numerical methods in fluids*, 51(4):439–468, 2006. (Cited on pages 9, 11 and 22)

[Röh11] D. Röhm. *Lattice boltzmann simulations on gpus*. Ph.D. thesis, Citeseer, 2011. (Cited on page 17)

[SAK02] T. Simonson, G. Archontis, M. Karplus. Free energy simulations come of age: protein- ligand recognition. *Accounts of chemical research*, 35(6):430–437, 2002. (Cited on page 25)

[Sch73] P. Schofield. Computer simulation studies of the liquid state. *Computer Physics Communications*, 5(1):17–23, 1973. (Cited on page 28)

[Sch08] U. D. Schiller. *Thermal fluctuations and boundary conditions in the lattice Boltzmann method*. Ph.D. thesis, Johannes Gutenberg-Universität, Mainz, 2008. (Cited on pages 11, 17, 20 and 21)

[SFCW13] R. Salomon-Ferrer, D. A. Case, R. C. Walker. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013. (Cited on page 25)

[SH98] X. Shan, X. He. Discretization of the velocity space in the solution of the Boltzmann equation. *Physical Review Letters*, 80(1):65, 1998. (Cited on page 20)

[SO99] Y. Sugita, Y. Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical physics letters*, 314(1):141–151, 1999. (Cited on page 25)

[SS17] D. Sean, G. W. Slater. Highly driven polymer translocation from a cylindrical cavity with a finite length. *The Journal of Chemical Physics*, 146(5):054903, 2017. (Cited on page 11)

[TFK06] J. Tölke, S. Freudiger, M. Krafczyk. An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations. *Computers & Fluids*, 35(8):820–830, 2006. (Cited on page 9)

[WIS10] A. Wiegmann, O. Iliev, A. Schindelin. Computer Aided Engineering of Filter Materials and Pleated Filters. 2010. (Cited on page 9)

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature