ECCOMAS Congress 2016
VII European Congress on Computational Methods in Applied Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris (eds.)
Crete Island, Greece, 5–10 June 2016

TOWARDS LATTICE-BOLTZMANN ON DYNAMICALLY ADAPTIVE GRIDS — MINIMALLY-INVASIVE GRID EXCHANGE IN ESPRESSO

Michael Lahnert[1], Carsten Burstedde[2], Christian Holm[3], Miriam Mehl[1], Georg Rempfer[3], and Florian Weik[3]

[1]Institute of Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
email: {michael.lahnert, miriam.mehl}@ipvs.uni-stuttgart.de
[2]Institute for Numerical Simulation and Hausdorff Center for Mathematics, University of Bonn
Wegelerstraße 6, 53115 Bonn, Germany
email: burstedde@ins.uni-bonn.de
[3]Institute for Computational Physics, University of Stuttgart
Allmandring 3, 70569 Stuttgart, Germany
email: {christian.holm, georg.rempfer, florian.weik}@icp.uni-stuttgart.de

Keywords: Octree Grids, Lattice-Boltzmann, Dynamical Adaptivity, Fluid Dynamics, Coupled Soft-Matter Simulations.

Abstract. *We present the minimally-invasive exchange of the regular Cartesian grid in the lattice-Boltzmann solver of ESPResSo by a dynamically-adaptive octree grid. Octree grids are favoured by computer scientists over other grid types as they are very memory-efficient. In addition, they represent a natural generalisation of regular Cartesian grids, such that most discretisation details of a regular grid solver can be maintained. Optimised codes, however, require a special tree-oriented grid traversal, which typically conflicts with existing simulation codes using various iterators, some for only parts of the grid, e.g., boundaries. ESPResSo is a large software package developed for soft-matter simulations involving fluid flow, electrostatic, and electrokinetic effects, and molecular dynamics. The currently used regular Cartesian grid hinders the simulation of realistic domain sizes and significant time periods, a problem that can be solved using grid adaptivity. In a first step, we focus on the lattice-Boltzmann flow solver in ESPResSo.*

`p4est` is a grid framework, that already provides dynamically adaptive quadtree and octree grids together with high-level interfaces for flexible grid traversals with direct neighbour access in all grid components. In this paper, we first describe extensions of `p4est` that were necessary to fulfill certain application requirements. The second part of our work consists of the minimally-invasive changes in ESPResSo preserving the expertise accumulated in the software's implementation over years. Our numerical results demonstrate physical correctness of the implementation, good parallel scalability and low overhead of the dynamical grid adaptivity. These are prerequisites to actually profit from grid adaptivity in terms of being able to simulate larger domains over longer time periods with limited computational resources. Thus, the current status forms a solid basis for further steps such as the development of refinement criteria, the setup of more realistic application scenarios, and a GPU implementation.

1 INTRODUCTION

Many scientifically and industrially relevant soft matter systems contain hydrodynamic flow with strong gradients in small subvolumes. Most biomolecules, including DNA and proteins, are charged when solvated in aqueous solution. Any charged object in electrolytic solution is surrounded by a so-called double layer – a thin layer, typically on the scale of one to a few hundred nanometres. This double layer is the only part of the volume that is not net neutral, a state maintained by the attraction of counterions and the repulsion of coions from the charged surface. Applying an external electric field causes fluid stress in the double layer and perturbs the ionic distributions. The fluid flow resulting from this complex coupling of ion and fluid dynamics is called electroosmotic flow (EOF) in the case of a charged surface at rest, while the resulting motion of a charged particle is called electrophoresis.

Both EOF and electrophoresis are important in applications such as the separation of biomolecules and other charged particles, DNA profiling, and pumping and sample transport in microfluidic devices, to name just a few.

Simulations of EOF and electrophoresis prove challenging, due to the large discrepancy between the nanometre scale of the double layer and the system sizes, which often range up to several hundred microns. Existing numerical schemes that work on unstructured grids like the finite-element method (FEM) or the finite-volume method (FVM) are great tools to investigate stationary electrophoresis and EOF systems, but the overhead incurred through remeshing makes these methods infeasible for systems containing mobile particles.

In this paper, we present our approach to moving a large simulation application from a regular Cartesian mesh to fully adaptive tree-structured Cartesian meshes in a minimally-invasive way. The application is the molecular and continuum dynamics package ESPResSo [1, 2], where we particularly focus on the included lattice-Boltzmann solver. For scalable parallel adaptive mesh refinement, we base our work on the software library `p4est` [3, 4].

Many large simulation codes use regular Cartesian meshes due to their cheap representation in memory, the simple data layout, the easy domain partitioning, and the straight-forward discretisations. However, the missing grid adaptivity in time and space limits the size of the domain and the simulation time accessible with reasonable computation time on available machines. Grid adaptivity would allow for coarsening of the mesh in regions of minor importance and provide the possibility for space-time adaptivity, i.e., to also apply larger time steps in spatially coarser domains. Two major types of adaptive meshes are widely used in simulation codes. The first type, unstructured meshes, offers a very high flexibility in terms of local mesh resolution, shape and even type of mesh elements. Thus, even very complex geometries can be represented accurately. However, unstructured meshes come with high memory requirements, and structures and element types completely different from regular Cartesian meshes. The second type of adaptive meshes consists of tree-structured Cartesian meshes, with octrees and quadtrees being the most famous examples. These meshes form a natural generalisation of regularly refined Cartesian meshes. They are constructed by recursively refining mesh elements into a fixed number of equally sized and equally shaped child elements. They are very efficient, in particular in terms of memory requirements and memory access patterns, if represented in a linearised form according to a depth-first traversal of the underlying tree and an ordering of the children prescribed by a space-filling curve [5, 6, 7]. Depending on the aggressiveness of the encoding, however, this may impede traversal of the mesh or parts thereof in an arbitrary order. In contrast, existing application codes on regularly refined Cartesian meshes often rely on access to parts of the mesh (e.g., boundaries), particular mesh regions, or neighbours of

mesh elements. Thus, integrating a tree-structured Cartesian mesh with an existing simulation code based on regular Cartesian meshes in a minimally-invasive way calls for a mesh interface that allows for the easy implementation of user-defined iterators and direct neighbour access. The challenge in this lies in finding the optimal trade-off between minimally-invasiveness, and memory and run-time efficiency.

Our idea for making spatial adaptivity more accessible to researchers from other fields is to turn the integration of an octree-based Cartesian grid into a three-step process. In a first step the regular grid of the existing application is replaced with an octree-based one, specialised for uniform refinement. In this way we are able to reproduce results of the existing implementation using a regular grid, without having to worry about spatial adaptivity, yet. This step preserves all of the regular grid's data-access patterns, up to a permutation of the mesh elements. This permutation is induced by the octree refinement, which replaces a traditional lexicographic ordering with one governed by the space-filling curve. Reordering this way benefits data locality and cache performance of large scale simulations, while the mathematical problem being solved remains the same. In case of a traversal of subregions of the grid, e.g. for setting boundary values, we may traverse the whole grid along the space-filling curve and only take action on the cells that meet a selection criterion, possibly caching the subset indices for later reuse. Alternatively, we may access the position of an initial cell and traverse the local part of the mesh by hopping to neighbours, independently of the space-filling curve.

The next step of our three-step process consists of solving numerical issues that arise when actually making use of the spatial adaptivity. An example for such an issue is the speed of sound in the lattice-Boltzmann fluid, which depends on the element size and the time step, and is therefore no longer uniform for a locally refined mesh with a constant time step. A uniform speed of sound can be recovered by using different time steps for differently refined parts of the mesh. There are several ways to consistently implement this, e.g., using a finite-volume approach [8], compact interpolation [9], or volumetric formulation [10].

As the final step, the application should be tuned by adapting both data access patterns and grid-traversal to make full use of the locality provided by the data storage scheme.

The need to be able to traverse subregions in our grid efficiently in step one introduces two requirements for the implementation of the grid library. First, we need random access to cells, and secondly, we need to be able to compute cell positions in our linearised array (as it is given by the space-filling curve) based on geometric coordinates. We will show that those two criteria are met by the `p4est_mesh` interface.

In the remainder of the paper, we present the software package ESPResSo together with a short introduction to the implemented lattice-Boltzmann solver on regular Cartesian meshes (Sect. 2), give an overview of the adaptive mesh software `p4est` (Sect. 3), a more detailed description of our contributions to both software packages establishing the first lattice-Boltzmann prototype on tree-structured meshes with minimal changes in ESPResSo (Sect. 4), followed by first results (Sect. 5) and a conclusion (Sect. 6).

2 ESPRESSO AND THE LATTICE-BOLTZMANN METHOD

2.1 ESPResSo

ESPResSo [1, 2], the extensible simulation package for research on soft matter systems, is a molecular dynamics (MD) program that focuses on coarse grained soft matter applications. While originally focused on the simulation of charged and uncharged polymers and colloids, today it implements a wide range of algorithms [11] for electrostatics [12, 13, 14, 15, 16], di-

electrics [17, 18, 19], magnetostatics [20] and hydrodynamics [11, 21], both for periodic and non-periodic systems. Applications include, amongst others, engineering problems such as soot aggregation [22], hydrogels [23], biological membranes [24], DNA like-charge attraction [25] and translocation [26], and ionic liquids [27]. Aside from the plethora of implemented methods, the major strength of ESPResSo lies in its ability to flexibly combine different methods and particle interactions to encompass new applications and systems quickly. ESPResSo is MPI parallelized and can employ GPU acceleration for hydrodynamics [21] and electrostatics calculations.

The regular mesh refinement currently used for continuum methods in ESPResSo hinders the efficient exploitation of different scales in different regions of the simulation volume, such that the generalisation to adaptively refined meshes is crucial for accurate simulations encompassing experimentally relevant time scales.

2.2 Lattice-Boltzmann

Lattice-Boltzmann (LB) is a method for solving the Navier-Stokes equations and similar partial differential equations, which was first proposed in the context of lattice gas automata. It was later discovered that the lattice-Boltzmann equation can also formally be derived as a discrete lattice approximation to the continuum Boltzmann equation for fluids, providing a sound mathematical basis for the method [28]. Further analysis based on the Chapman-Enskog expansion shows that the LB equation is equivalent to the full Navier-Stokes equations in the limit of low Mach numbers. Dünweg and Ladd gave a comprehensive analysis of the LB method and many of its extensions relevant to soft matter simulations in 2009 [29]. A good overview of the state of the development and its possible applications were given by Succi, Sbragaglia, and Ubertini in 2010 [30].

The LB equation is a discretisation of the Boltzmann equation. The Boltzmann equation is defined in the seven-dimensional space with three spatial coordinates, three velocity or momentum components and time as independent variables and describes the time evolution of the microscopic particle distribution. The lattice-Boltzmann method discretises the velocity space with a finite number of discrete velocities $\boldsymbol{c}_i$ and finite differences on a cubic grid in the three-dimensional coordinate space. This results in the lattice-Boltzmann equation (LBE)

$$f_i(\boldsymbol{r} + \boldsymbol{c}_i \Delta t, t + \Delta t) = f_i(\boldsymbol{r}, t) + \sum_k L_{ik} \left\{ f_k^{\mathrm{eq}} - f_k(\boldsymbol{r}, t) \right\} + \Psi_i \ , \tag{1}$$

where f_i denotes the densities of virtual particle populations assuming the velocity $\boldsymbol{c}_i$, and Ψ_i models the density change of the population f_i due to external forces. The discrete velocities are chosen such that they transport particles from one lattice node to one of its neighbours in one time step. Fig. 1 shows the widely used D3Q19 discretisation for the velocities with 19 velocity components associated to the centre of a three-dimensional cubic grid cell. We assume linear relaxation towards the equilibrium distributions f_i^{eq}, according to the linear operator L_{ik}.

The macroscopic values for the density ρ, momentum $\boldsymbol{p}$, and stress $\overline{\boldsymbol{\pi}}$ can be recovered from the microscopic particle populations through

$$\rho = \sum_i f_i, \qquad \boldsymbol{p} = \sum_i f_i \boldsymbol{c}_i, \qquad \overline{\boldsymbol{\pi}} = \sum_i f_i \boldsymbol{c}_i \otimes \boldsymbol{c}_i, \tag{2}$$

where $\otimes$ denotes the tensor product. Using those, the lattice equilibrium distribution is given

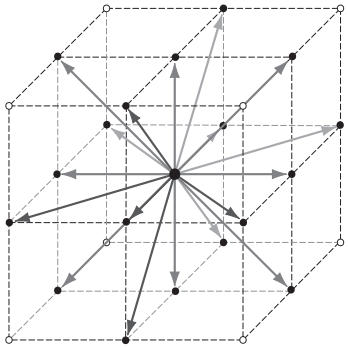

Figure 1: One cell from a D3Q19 LB grid, depicting the 19 discrete velocities, connecting the node with itself, its nearest, and next nearest neighbours. Image adapted from [31].

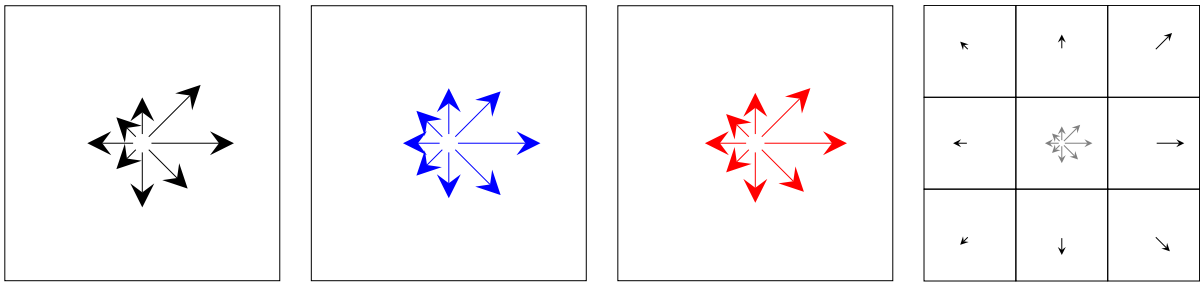

Figure 2: Schematic view of the collision and the streaming step. The arrows indicate the discrete velocities $\boldsymbol{c}_i$ in a two-dimensional quadratic grid cell scaled with the respective densities f_i. (a) – (c): collision step. Left to right: pre-collision populations, virtual local equilibrium, post-collision populations. (d): streaming step. Grey arrows indicate the pre-streaming population in the centre cell, black arrows the same populations transported to the respective neighbour cells, i.e., after the streaming step.

by a second order expansion of the continuum Maxwell-Boltzmann equilibrium distribution

$$f_i^{\mathrm{eq}} = a_i \rho \left(1 + \frac{\boldsymbol{v} \cdot \boldsymbol{c}_i}{c_s^2} + \frac{(\boldsymbol{v} \cdot \boldsymbol{c}_i)^2}{2c_s^4} - \frac{v^2}{2c_s^2} \right) , \tag{3}$$

with prefactors a_i and the speed of sound c_s depending on the specific lattice geometry.

Algorithmically, performing a time step for the LBE (1) consists of two parts with different requirements in terms of data access: Whereas the collision is a local operation modelling the interaction of densities of different velocity within a local cell described by the operator L_{ik}, the streaming step transports densities into neighbouring cells according to the direction of the respective velocity $\boldsymbol{c}_i$. Fig. 2 schematically shows the collision step as a purely cell-local operation on population densities and the equilibrium densities. In addition, it displays the streaming as a transport of data from one cell to its neighbour cells.

Collision. ESPResSo implicitly implements different versions of the collision operator: The simplest choice for the collision operator $L_{ik} = \lambda \delta_{ik}$ is the so-called BGK collision operator (after the inventors Bhatnagar, Gross, and Krook) using only a single relaxation parameter λ for all populations, which can be chosen such that the fluid assumes the desired shear viscosity. Other fluid properties such as the bulk viscosity cannot be chosen independently. The full multi-relaxation time LB method (MRT-LB) results in less limitations. Here one relaxes different linear combinations m_i of the velocity populations f_i towards their equilibrium values using

different relaxation rates λ_i. These linear combinations correspond to the different macroscopic quantities. The MRT-LB forms a superset of the simpler collision operators, i.e. by choosing the λ_i accordingly, the MRT collision operator can reproduce the simpler BGK and TRT (two relaxation time) collision operators [32].

Streaming. In terms of streaming, it is important to note that, due to the requirement that virtual particle populations are transported exactly from one cell midpoint to the midpoint of the respective neighbour cell, the size of the time step Δt is proportional to the side length of the grid cells. This observation plays a major role in adaptively refined grids in Sect. 4.

By identifying nodes on the boundaries of the cubic grid with the corresponding nodes on the opposite side, one can trivially implement periodic boundary conditions in the streaming step. Other boundary conditions, such as fixed velocity or fixed pressure are more complex to implement. For simplicity, we will only discuss no-slip boundary conditions here. Further information can be found in [33]. One simple way to realise no-slip boundary conditions is by introducing so-called bounce-back nodes in the grid. Instead of streaming populations along their respective velocities, they reverse the velocities of the populations and transfer them back to the originating node. This procedure mimics a fluid flow on the opposite side of the boundary, which is antisymmetric to the real one. Due to the antisymmetry, the flow velocity on the boundary needs to vanish.

2.3 Lattice-Boltzmann in ESPResSo

While ESPResSo is primarily a molecular dynamics code, it includes an implementation of the multi-relaxation time D3Q19 lattice-Boltzmann method (MRT LB) [34], to solve the fluctuating Navier-Stokes equations. This lattice-Boltzmann fluid can be point-coupled to MD particles using the method published by Dünweg and Ladd [29]. The LB is MPI parallelized, which is realised by domain decomposition and halo communication at the node boundaries.

3 PARALLEL ADAPTIVE TREE-STRUCTURED GRIDS AND P4EST

3.1 Parallel Tree-Structured Grids

Tree-structured Cartesian grids are a natural generalisation of regular Cartesian grids that have two important advantages over unstructured meshes: 1) The shape of the grid elements is the same as in a regular Cartesian mesh. Thus, the underlying discretisation of a solver operating on a regular Cartesian mesh can be generalised to this type of adaptive grid by locally incorporating interpolation and projection on hanging nodes and/or faces [35, 36, 37]. 2) The grids are strictly structured, which allows for a very cheap representation in memory and for efficient grid partitioning. The construction principle of a tree-structured grid is the recursive refinement of grid cells into a fixed number of children of equal size and shape. If the refinement is performed globally, we get a regular Cartesian mesh, if we refine only locally in certain marked cells, we get an adaptively refined grid. Figure 3 shows a pair of two-dimensional quadtree grids (refinement of each quad into four subquads) and their respective cell trees.

In order to provide a unique ordering of grid cells on all levels, discrete iterates of space-filling-curves are widely used. There are space-filling curves for two-dimensional and for three-dimensional tree-structured grids [38] (in fact, many generalise to arbitrary dimension) and for different refinement strategies. Figure 4 shows the iterate of a Peano-curve in a two-dimensional grid with a recursive refinement of grid cells into nine children, a Hilbert-curve iterate [39] in a quadtree grid, and the Z-curve or Morton-order [40] in a quadtree grid.

The tree-structure in combination with a space-filling curve iterate allows for a very efficient

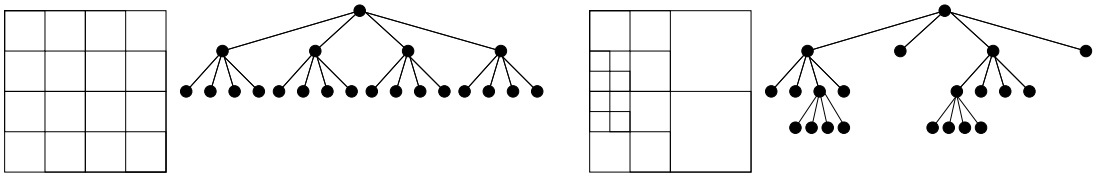

Figure 3: Left: a regularly refined quadtree grid of refinement level two. Right: an adaptively refined quadtree grid of minimal refinement level one and maximal refinement level three. In both cases, we also display the corresponding tree structure.

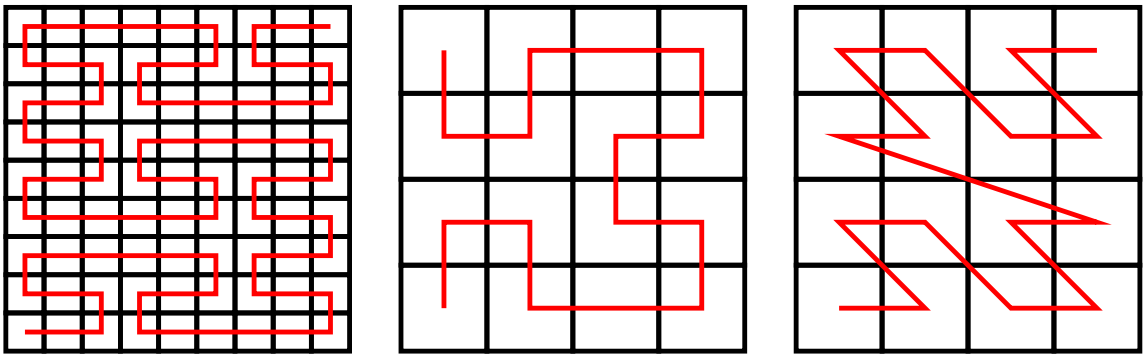

Figure 4: Some exemplary space-filling curve iterates on grids with regular refinement of level two. Left: Peano-curve; middle: Hilbert-curve; right: Z-curve or Morton-order.

linearised bit-code representation of the whole grid. This representation follows a depth-first grid traversal with an ordering of children according to the chosen space-filling curve. A '1' indicates a refined cell, whereas a '0' represents an unrefined cell. Figure 5 shows a two-dimensional adaptive quadtree mesh ordered according to the Z-curve together with the cell tree and the linearised representation.

In scientific simulation software, the combination of tree-structured grids and space-filling curves has been used in several ways, for example augmented by hashing [41], or for partial differential equation solvers with cache-optimised data administration [42, 5]. `Octor` [43] and `Dendro` [6] are two examples of parallel octree libraries that have been scaled to 62,000 [44] and 32,000 [45] cores, operating on parent-child pointers and a linearised octant storage, respectively.

3.2 The `p4est` Software

The `p4est` software library [3] extends the linearised storage strategy to a forest of interconnected octrees [46, 47]. Figure 6 shows a forest of two trees as an example. `p4est` provides

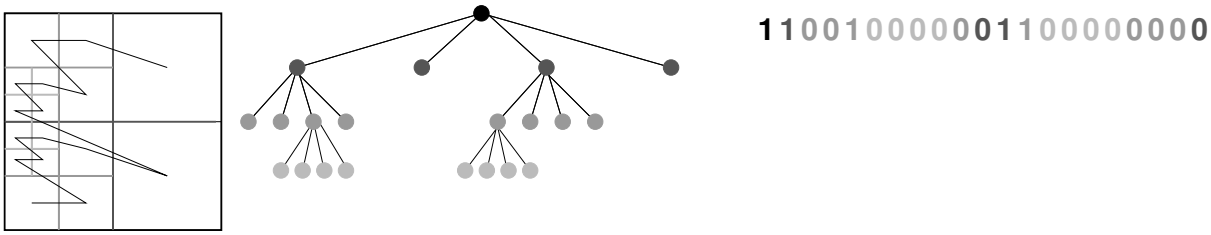

Figure 5: Example for an adaptively refined quadtree grid with cell-ordering according to the Z-curve, the corresponding cell-tree and the linearised bit-code.

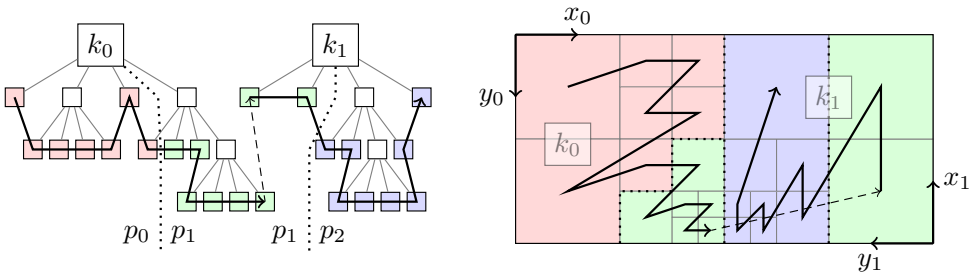

Figure 6: An example 2D forest of two adaptive quadtrees. It is partitioned into three processors (colour-coded). Note that the partition boundary does not always coincide with a tree boundary and that the coordinate systems of the trees can be rotated or mirrored relative to each other. This mesh is 2:1 balanced: Neighbour quadrants differ in size by at most a factor of two. We use the terms quadrant/octant, leaf, and element as synonyms, since they are logically equivalent.

a publicly available implementation of parallel algorithms required to create and dynamically refine, coarsen, and repartition an adaptive mesh. Furthermore, `p4est` includes an efficient 2:1 balancing algorithm [48]. In our context, 2:1 balancing means that the refinement is graded in such a way that it has a maximal level difference of one between adjacent grid cells. This feature limits the number of possible configurations of different-sized neighbours that can occur with arbitrary adaptation. Some applications are designed not to use this feature [49], while we exploit it in the lattice-Boltzmann code to maximise the simplicity of discretisation and time stepping.

`p4est` manages the dynamic-adaptive connectivity of the mesh elements. It is a C library using the MPI standard for parallelisation. The "root" or "coarse" mesh is a collection of tree roots, each identified with its cubical subvolume of the domain, and augmented by the connectivity relation between neighbouring trees. For many applications, it is sufficient to call builtin functions that create a one-tree unit cube or a rectangular brick of a small number of trees. The initial refinement of the forest is created by calling the New function with a given coarse mesh and an initial level to create a uniform distributed mesh. The refinement and coarsening functions can then be invoked any number of times to locally swap mesh elements with their children and vice versa.

`p4est` uses the Morton curve to order the elements in each tree; see Figures 4 (right) and 6. The code stores just the leaves of the forest by their lower left corner and refinement level and operates on the mesh by exploiting the mathematical structure implicit in the curve. Refinement and coarsening are local in the sense that a parent occupies the same segment in a curve as its four (2D) or eight (3D) child elements. The Partition algorithm exploits this property to shift the elements between processors without changing their order. Optionally, each element may be assigned a weight to account for a varying computational load throughout the mesh.

The basic grid traversal component of `p4est` is the `p4est_iterate` routine [50] that visits all interfaces of local mesh cells across all co-dimensions. The user may define callback functions separately for each co-dimension. Effectively, each mesh entity (cell, face, edge, or corner) is visited exactly once, and the callback function is informed with a list of the mesh cells adjacent to the visited entity. In this sense, the iterator provides a view of both the primal and the dual mesh local to a process. Calling `p4est_iterate` is the most general and lightweight way to expose the mesh connectivity to applications, formulated exclusively in `p4est`'s encoding of mesh entities and their relative orientations.

Existing simulation codes often require direct (random) access to neighbours of cells or cor-

ners, ideally in constant time. To meet this need, `p4est` provides the so-called `p4est_mesh` that we will shortly describe in more detail. In fact, we have chosen it as the central component for the integration of `p4est` with ESPResSo.

The `p4est_mesh` data structure consists of several lookup tables that store neighbourhood information for each grid cell as well as level-wise cell lists. It covers all process-local cells and the direct off-processor neighbour cells (ghosts). Thus, this module provides a way to traverse the local partition of the grid independently of the space-filling curve and allows for the evaluation of arbitrary stencils. However, this flexibility comes at the cost of additional run-time (it is created by calling `p4est_iterate` internally) and memory (the lookup tables are allocated), cf. section 5.3. This overhead can be amortised if the mesh structure is queried repeatedly, since each lookup only requires $\mathcal{O}(1)$ time. Information on a neighbouring cell is encoded in two integers: The first stores the index of the neighbour cell in the global cell list of the corresponding processor, the second stores the size and relative orientation of the neighbour cell. The latter is important if the neighbour cell belongs to a different quadtree or octree, which might have a different orientation. Due to the 2:1 balancing that we invoke, there are only three different cases: the neighbour of a cell can have the same size or be either twice or half as big. In the latter case, we store the indices of all smaller neighbours in the respective direction.

In summary, `p4est` provides a concise set of parallel AMR algorithms that are tailored to dynamic-adaptive PDE solution, but can also be used more generally whenever an efficient parallel space partition is required. `p4est` has been shown to scale to over 458k cores [50], with applications using it successfully on 1.57M cores [51] and 3.14M cores [52].

4 LBM ON SPATIALLY-ADAPTIVE GRIDS

In this section, we present the exchange of the regular Cartesian grid in ESPResSo by a dynamically-adaptive tree-structured grid provided by `p4est`. In Section 4.1, the underlying ideas of the lattice-Boltzmann solver on tree-structured grids are introduced. This includes, in particular, the realisation of collision and streaming, as well as the adaptive time stepping at boundaries between different spatial refinement levels of the octree grid. These discretisation details, together with the velocity discretisation scheme, define the neighbour cells that need to be made accessible in the `p4est_mesh`. The adaptations to `p4est`, including the generation of the `p4est_mesh` for our adaptive lattice-Boltzmann implementation is outlined in Section 4.2, whereas Section 4.3 sketches the changes in ESPResSo that were required for the replacement of the original grid implementation by the `p4est` grid.

4.1 Adaptive Formulation of the LBM

In order to cope with different mesh widths over the computational domain imposed by spatial adaptivity, the classical algorithm of the LBM needs to be extended. We use the collision operator independent scheme proposed by Rohde, Kandhai, Derksen, and van den Akker in 2006 [53]. The scheme proposes a volumetric formulation of the LBM, i.e., the nodes containing the populations are associated to the cell centres.

The algorithm consists of several steps that are shown schematically in Fig. 7 for a part of the mesh consisting of a coarse grid cell and a further refined neighbour cell with four children. Outward facing arrows indicate streaming, inward facing arrows collision. The steps of the algorithm result from two basic observations: 1) Due to the proportionality of the time step to the cell width, we have to perform two time steps in the smaller cells, while performing only one (twice as large) time step in the coarser cell. We will use the optional feature of

	coarse	fine
Mesh width Δx	Δx_c	$\Delta x_f = 0.5 \cdot \Delta x_c$
Time step Δt	Δt_c	$\Delta t_f = 0.5 \cdot \Delta t_c$
Lattice velocities c_i	$c_{i,c}$	$c_{i,f} = c_{i,c}$
Viscosity ν	ν_c	$\nu_f = \nu_c$
Relaxation parameter λ (BGK)	λ_c	$\lambda_f = 2.5 \cdot \lambda_c + 0.5$
Velocity u	u_c	u_f
External acceleration/force g	g_c	$g_f = 0.5 \cdot g_c$

Table 1: Conversion of parameters and observables between fine and coarse cells for 2:1 balanced grids according to [53].

`p4est_partition` to assign different weights to the leaves for load balancing. 2) Streaming populations from the finer grid cells into the coarse cell and vice versa requires populations associated to a common grid level. Thus, we have to interpolate populations of the coarse cell to virtual child cell populations before the streaming step and restrict the results back to the coarse cell after the second time step of the fine cells. Additionally, some fluid properties differ between different discretisation widths as shown in Table 1.

Step 1 in Fig. 7 depicts the collision on both coarse and fine cells for the discussed example situation. After this collision, we provide streaming partners for the finer cells by copying the coarse cell populations to four (2D) or eight (3D) virtual child cells (step 2). Then a streaming step can be performed on both the virtual and the original fine grid cells (step 3). Note that in the virtual fine cells, only populations associated to velocities pointing towards the original fine grid cells actually have to be streamed. In step 4a, the second collision step in the original fine cells is executed, followed by the second fine grid streaming step (step 4b). Finally, we transfer the virtual fine grid populations back to the coarse grid by taking the arithmetic mean, and finish the coarse grid time step with the streaming of the coarse grid populations whose velocities point towards other coarse cells. Note that, in a more general case without 2:1 balancing, we would have to perform more substeps on the fine grid cells before returning to the coarse level.

The generalisation of the presented adaptive lattice-Boltzmann algorithm to three-dimensional 2:1 balanced octree grids is, in principle, straight-forward. If we summarise the requirements for the octree mesh stemming from the adaptive LBM, we can state that the `p4est_mesh` has to provide direct access to the 18 neighbour cells corresponding to the 19 velocity directions minus the zero velocity in the D3Q19 LBM (compare Fig. 1). In addition, we have to implement the realisation of the virtual fine grid cells in `p4est`, a feature that was not available previously.

4.2 Preparing `p4est` for adaptive lattice-Boltzmann

As mentioned above, two aspects in `p4est` are crucial for a successful and minimally-invasive integration of the `p4est` grid in the lattice-Boltzmann solver in ESPResSo: the provision of the required neighbour cells in `p4est_mesh` and the implementation of virtual children of grid cells neighbouring a further refined region. In this section, we describe the generation of the `p4est_mesh` in more detail and briefly sketch the implementation of virtual refinement.

`p4est_mesh`. The first issue in generating the `p4est_mesh` (during a grid traversal using `p4est_iterate`) is to identify the neighbours actually required by the LBM among the information that can be generated with `p4est` callback functions for neighbour detection of faces, edges and corners. The iterator provides a top-view on each element of each co-dimension, i.e., two adjacent cells for faces, four for edges, and eight for corners. In an adaptive mesh, the in-

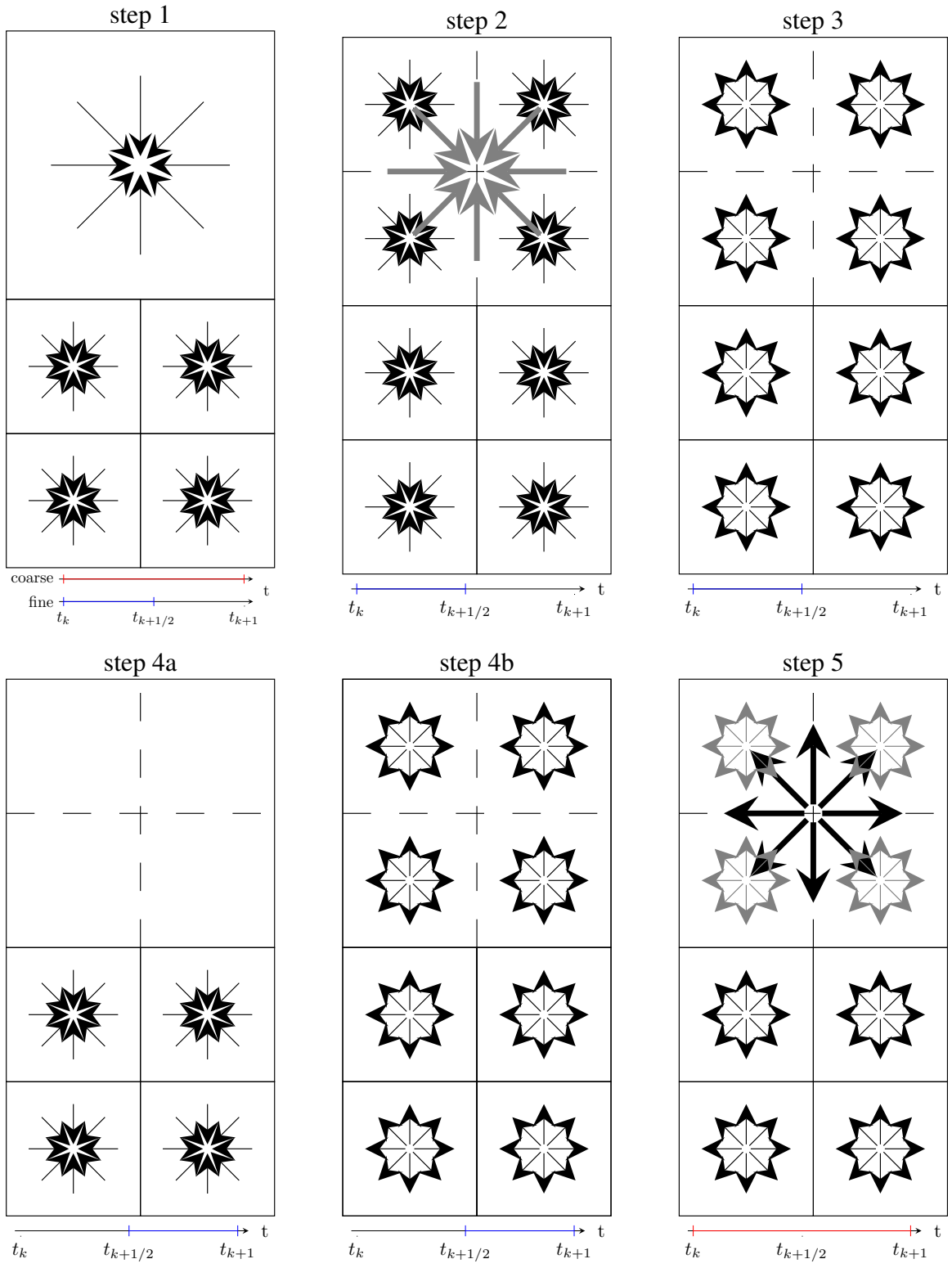

Figure 7: Collision operator independent extension to the algorithm of the LBM to spatially adaptive Cartesian grids as proposed by [53]. 1. Collision on coarse and fine grid. 2. Homogeneous redistribution of populations from coarse to fine grid. 3. Streaming step on virtual fine and fine grid. 4a. Collision step in fine grid. 4b. Streaming step on virtual fine and fine grid. 5. Homogeneous redistribution of populations from virtual fine to coarse grid. Note that steps 4a and 4b are only performed once due to the 2:1 balancing constraint.

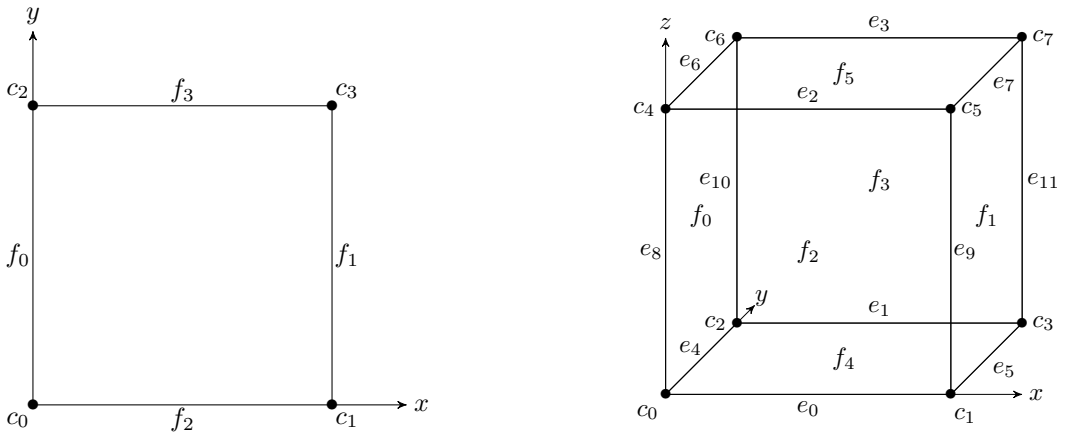

Figure 8: Naming scheme of faces, edges, and corners in `p4est`.

terface may be hanging and may have one side consisting of four small faces instead of one, or two small edges instead of one. We display two possible situations in Figure 9. We have to combine this information to end up with the list of the 18 neighbours required in the D3Q19 LBM, which means that we have to eliminate face neighbours from the edge neighbour list and face and edge neighbours from the corner neighbour list. The Z-curve allows for a very simple and efficient calculation of the Morton-index of neighbour cells. However, since the grid is adaptively refined, the Morton index is not the same as the position in the actual cell list. Therefore, neighbour cells are detected via a recursive divide-and-conquer algorithm operating on the cell list[1]. `p4est` uses a fixed naming scheme for faces, edges and corners of a cell as displayed in Figure 8. This helps to reduce the costs for the neighbour search by allowing us to fill the neighbour lookup tables pairwise for neighbour cells within the same tree using a simple mechanism based on XOR to determine the naming of the respective face, edge or corner in the considered partner cell.

For the detection of neighbours across boundaries between different octrees, the relative orientation of the involved octrees has to be taken into account. Figure 9 shows two examples for such situations. We observe that, in this case, even the number of neighbours of an edge or corner can take any value. However, for our LBM, only rectangular meshes are envisioned such that at least this latter issue does not occur. As different trees may have different local coordinate systems we cannot know beforehand which edge or corner index the diagonally opposite element will have. The orientation and the index of a neighbouring tree can be retrieved from the `p4est_connectivity`, which represents the macroscopic structure of the grid, i.e., the way that the different octrees are arranged relative to each other. With this, we calculate the edge or corner indices of neighbours using a transformation.

In order to transform the index of a corner from the perspective of one tree to the perspective of another across a face, a transformation was proposed in [3] that only needs the face indices of the common face from both perspectives (f_i, f_j), the relative orientation (r) of the two octree faces and the respective face corner ξ_i. A local Morton order is imposed on the corners of each face by enumerating all four in ascending order w.r.t. `p4est`'s naming scheme. Then the face

[1] `p4est` works with an optimised binary search that recursively splits the list of the leaf cells covered by a branch of the octree into eight sublists, one for each child branch, during the top-down depth-first tree traversal in `p4est_iterate` [50]. This function also identifies neighbours of different sizes created by adaptation.

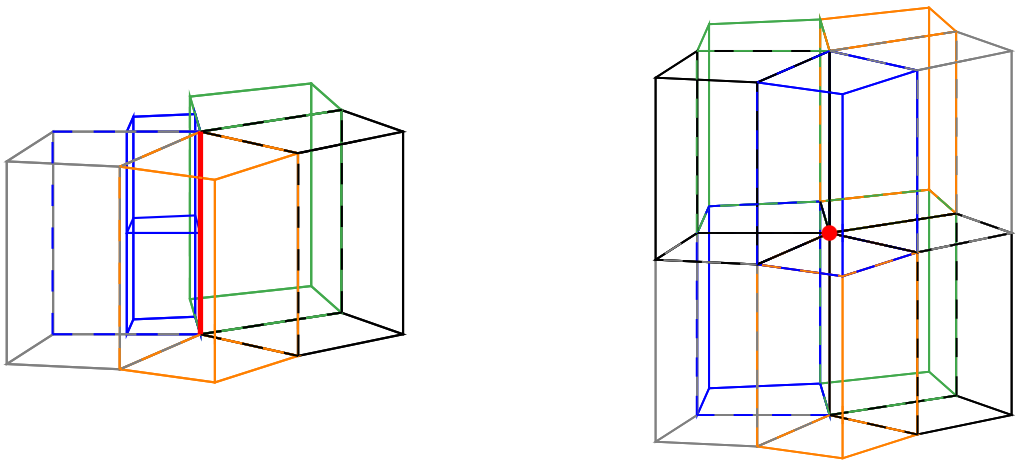

Figure 9: Exemplary multiple tree scenarios to find edge or corner neighbours for each cell. Each tree is drawn in a different colour. The element over which neighbours are to be found is marked in red.

corner number ξ_j viewed from the adjacent octree is given by the permutation

$$\xi_j = \mathcal{P}_c(\mathcal{Q}_c(\mathcal{R}_c(f_i, f_j), r), \xi_i) \equiv \xi_j(\xi_i) \tag{4}$$

with

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

The permutations of the face corners listed in the rows of $\mathcal{P}_c$ correspond to the 2×4 elements of the dihedral group of face orientations. The matrices $\mathcal{R}_c$ and $\mathcal{Q}_c$ are designed to choose the correct one in a non-redundant way (using 0-based indexing).

To extend this logic to edges, we propose a similar transformation. In addition to the already defined face corners ξ_i, we define face edges ν_i in the same way, i.e. by enumerating the edges spanning a face in ascending (Morton-)order. Then each face edge is enclosed by two face corners that can be written as rows of

$$\Xi_i = \begin{pmatrix} 0 & 2 \\ 1 & 3 \\ 0 & 1 \\ 2 & 3 \end{pmatrix}. \tag{6}$$

We can thus infer the edge corners seen from the adjacent tree by the element-wise application of (4),

$$\Xi_j(\nu_i, k) = \xi_j\left(\Xi_i(\nu_i, k)\right), \quad k = 0, 1. \tag{7}$$

Using (6) in reverse, the resulting two face corners can be identified one-to-one with the face edge's number seen from the other tree. We can condense this process into the form used earlier by writing the edge transformation

$$\nu_j = \mathcal{P}_e(\mathcal{Q}_e(\mathcal{R}_e(f_i, f_j), r), \nu_i) \equiv \nu_j(\nu_i) \tag{8}$$

with

$$\mathcal{R}_e = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \mathcal{Q}_e = \begin{pmatrix} 4 & 1 & 2 & 7 \\ 0 & 6 & 5 & 3 \end{pmatrix} \quad \mathcal{P}_e = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 3 & 2 \\ 1 & 0 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 1 & 0 \\ 3 & 2 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}. \tag{9}$$

The two permutations (4) and (11) allow excluding face neighbouring cells when creating edges and corner lookup tables and are sufficient for our D3Q19 LBM.

To complete `p4est_mesh` we additionally have to exclude edge neighbours, when the corner lookup table is created. We define edge corners μ_i, analog to face corners ξ_i and face edges ν_i. It has to be noted that in this case the relative orientation is no longer between two faces but between two edges. Thus, we will distinguish the relative orientation between two edges from the relative orientation between two faces by denoting the former by s. In case of two faces only two cases have to be distinguished instead of four, because edges can either face the same or the opposite direction. Again, this information can be derived from the face corner transformation. Inspecting (5) and (7), we see that some transformations will lead to two ascending edge corner numbers and others to descending ones. In the latter case the edge is flipped when going between the two trees and the edge corners must be permuted.

To determine s for each edge neighbour we consider all three edges that contain the corner across which we want to find neighbours separately. Beginning with the current cell, we set the relative orientation $s = 0$. Then we can iterate over the two faces adjacent to the respective edge into the next cell. One of the two faces we do not have to consider, because it is the face across which we entered the cell. The relative orientation of the edge s in the face neighbouring cell is determined using (10).

$$s_{\text{new}} = (s_{\text{old}} + (\mathcal{P}_s(\mathcal{Q}_s(\mathcal{R}_s(f_i, f_j), r), \nu_i))) \mod 2 \tag{10}$$

with

$$\mathcal{R}_s = \mathcal{R}_e \quad \mathcal{Q}_s = \begin{pmatrix} 0 & 2 & 1 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \quad \mathcal{P}_s = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}. \tag{11}$$

To map edge corners from one octree to the other we can apply the transformation

$$\mu_j = \mathcal{P}_{ec}(\mathcal{Q}_{ec}(r), \mu_i) \tag{12}$$

with

$$\mathcal{Q}_{ec} = \begin{pmatrix} 0 & 1 \end{pmatrix} \quad \mathcal{P}_{ec} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{13}$$

In the implementation, the relative orientation s does not need to be calculated separately for each edge neighbour. Instead, s is stored inside of `p4est_connectivity`. This redundancy facilitates creating `p4est_mesh`, because the above transformation can be applied directly.

Summarising, the first permutation from [3] allows removing the face neighbours when setting up the lookup table for corner neighbours, while the third permutation allows to omit edge

neighbours in that case. The second permutation is used to remove face neighbours when setting up edge neighbour lookup tables.

Virtual refinements. Our adaptive LBM algorithm requires to persistently store the populations within virtually embedded cells over multiple grid traversals to restrict their results back to the coarse cell after two smaller time steps on the fine grid. Therefore, we have to store their data additionally to the data of the original coarse cell. This leads to two obvious options considering the question which cells we store in the list of octree leaf quadrants. 1) We store the virtual fine cells as part of the leaf list and create the parent cell as a virtual cell or 2) we store the coarse cell and create four (2D) or eight (3D) virtual fine cells. We choose the latter solution and store the virtual cells in Morton order using a modified version of the `p4est_childrен` inside `p4est_mesh`, creating them simultaneously with the lookup tables. This algorithm differs from a standard refinement step only in the sense that the resulting quadrants are not stored within the list of leaf quadrants and that the parent quadrant is not discarded. This allows integrating virtual cells into an existing application in a completely transparent way, hiding the whole process from the application via abstraction in the neighbour lookup. Furthermore, we can account for the increased load of those cells that contain virtual children by modifying the weight passed to the partition function [3].

4.3 Minimally-Invasive Integration of `p4est` with ESPResSo

As a very general change that is not directly related to grid adaptivity, we wrote some additional callbacks such that the manager node can call the slave nodes to perform the respective distributed routines of `p4est` for initialising, traversing or deallocating a grid. This enables `p4est`, which is designed as a fully distributed library, to work with ESPResSo, which uses a manager/worker architecture. In case of a manager/worker architecture the system is controlled by one or more manager nodes which distribute the work among the worker nodes. Note, that the number of worker nodes is generally significantly bigger than the number of manager nodes. In case of a fully distributed architecture all nodes are equal and have to agree on distribution of the workload. Therefore, a manager/worker scheme avoids solving some (either implicit or explicit) consensus problems at the cost of a reduced fault tolerance.

Besides that, the actual changes that were made to ESPResSo for integrating `p4est` were fairly small. We changed the storage scheme and the grid-traversal from the classical lexicographic ordering to an ordering along the space-filling curve. Cell-local functions, a category in which in case of the LBM all algorithmic steps belong except from the streaming step, could be preserved the way they were. We only changed the function headers to make the respective functions callable as volume callback function using `p4est_iterate` and, for efficiency reasons, changed the data layout from a propagation optimised storage scheme to a collision optimised data scheme, cf. [54]. In a collision optimised data layout, all populations of a cell are grouped locally in memory, i.e., the memory contains $N \times Q$ populations. On the other hand, in a population optimised data layout all cells of a lattice velocity $\boldsymbol{c_i}$ are grouped locally such that $Q \times N$ populations are stored. Both data layout schemes are illustrated in Fig. 10. The reason for switching the data layout is that, in case of dynamic refining and coarsening, we have to shift data in a collision optimised scheme once to preserve data locality as it is given by the space-filling curve. In a propagation optimised scheme, we have to shift data Q times, i.e., in our case 19 times. The remainder of the code was preserved.

For functions requiring access to neighbouring cell data, which in case of the LBM is the streaming step, we changed the header such that the function is callable as a volume callback function using `p4est_iterate`. Neighbour data are accessed using the lookup tables as they

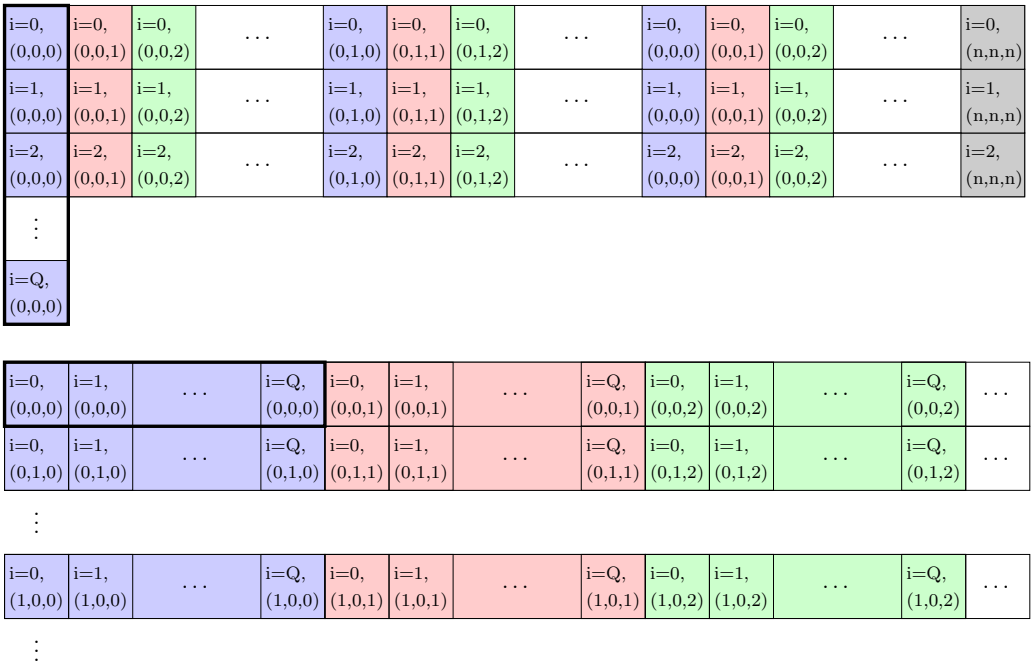

Figure 10: Comparison of different data layout schemes for the LBM. The highlighted black rectangle shows which data have to be loaded for the collision step in the first cell. The upper half shows the propagation optimised data layout which requires more cache lines to be read for the collision step. The lower half of the figure shows the collision optimised data layout. Image adapted from [33].

are provided by `p4est_mesh` instead of performing classical index calculations in the regular grid's (i, j, k)-indexing scheme. To hide the complexity of the different tables from the user, we extended the `p4est_mesh` module by a function returning a list of pointers to the respective neighbouring quadrants as well as a list of encodings specifying the meta information contained in the mesh.

Moreover, `p4est_iterate` does only visit local quadrants, specifically it does not visit ghost quadrants in the parallel domain partitioned version. That means we do not perform a collision step in the ghost layer but we perform an additional communication step after the collision step. This is motivated by a more rigorous understanding of ownership of quadrants: the data of a quadrant is only written by the process that logically owns this quadrant. Following to this idea and in order to perform correct streaming steps at process boundaries, the streaming and the bounce back step (at boundaries) have been slightly adapted as shown in Fig. 11. In case that a neighbouring cell is part of the ghost layer we mirror the streaming step. Instead of streaming populations from the current cell into the ghost cell, we do the inverse streaming operation (for the inverse lattice velocity) from the ghost cell into the current cell.

While the complete integration of spatial adaptivity is not yet finished, the mesh is already prepared for saving virtual quadrants. To achieve full dynamical adaptivity, we will change the grid traversal to level-wise traversals matching the time-adaptive algorithm outlined in 4.1 and Fig. 7 instead of traversing the entire grid while explicitly omitting all cells of different levels.

5 RESULTS

To test our integration we performed some tests on a shared memory machine with 72 physical cores (Intel® Xeon® CPU E7-8880 @2.30GHz) and 512 GB RAM. In 5.1, we compare our new `p4est` grid using a regular discretisation to the existing regular grid. To mimic spa-

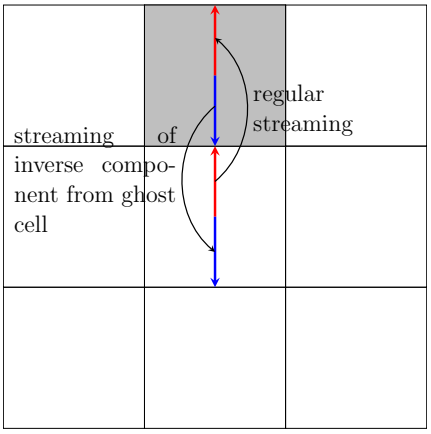

Figure 11: Illustration of the streaming step at neighbouring cells from the ghost layer (grey). Instead of streaming from the local cell into the ghost cell (red) we perform the inverse streaming step from the ghost cell to the current cell (blue).

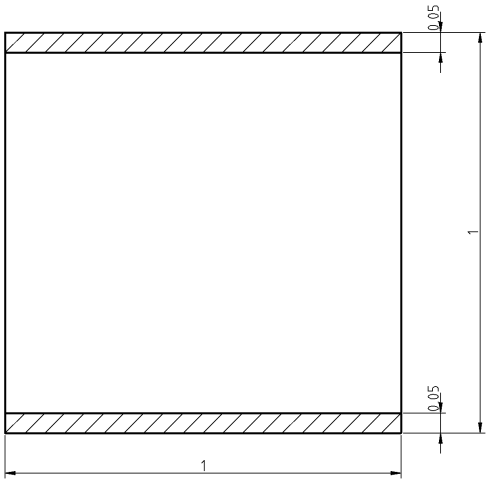

Figure 12: Geometry of the considered Poiseuille flow scenario. Note that the channel is not cylindrical but extruded to better match with the unit hypercube.

tial adaptivity in terms of realistic run-time, we do not use the information that the actually used grid is regularly refined and constant over time. Instead, we reconstruct the entries of `p4est_ghost` and `p4est_mesh` in each time step. In 5.2, we analyse the additional costs of this choice, i.e., the time it takes to create both structures. The section is concluded by comparing the run-time for traversing the grid using `p4est_iterate` and `p4est_mesh` in 5.3.

5.1 Run-time of LBM on Different Grids

To test our integration, we performed some scalability tests for a simple Poiseuille flow scenario as it is shown in Fig. 12. Note that the channel was not rotated to a cylindrical shape but that it has been extruded to the size of the unit cube, thus containing more fluid and fewer obstacle cells compared to a cylindrical channel.

We compared our `p4est` based implementation using a regularly refined octree and a collision optimised data layout to the existing regular grid with lexicographic cell-ordering using a propagation optimised data layout. For simulating the run-time effects of dynamic grid adaptivity, we did not use the information that we have both a constant layer of ghost cells

Proc	NUPS p4est					NUPS regular grid				
	Level 4	Level 5	Level 6	Level 7	Level 8	Level 4	Level 5	Level 6	Level 7	Level 8
16	1,323.85	2,824.1	3,621.2	4,044.3	4,300.2	2,185.7	9,019.6	11,083.8	13,934.5	13,867.5
32	1,411.93	4,567.6	6,450.7	7,513.4	8,294.2	2,200.96	11,583	21,357.7	23,125	27,104
64	1,422.23	6,261.7	9,461.7	13,600.2	15,498.8	1,895.42	11,279.9	36,880	39,871	46,580

Table 2: Comparison of lattice-Boltzmann run-times using the p4est grid with existing regular grids on various discretisation levels for an increasing number of MPI processes in NUPS (node updates per second).

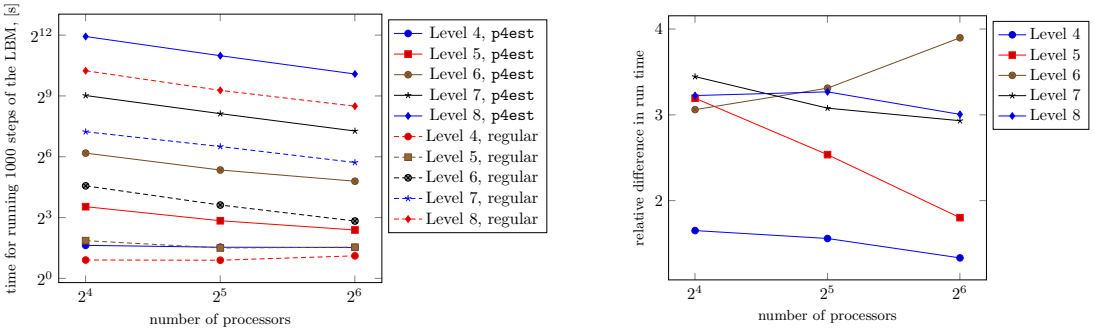

Figure 13: Comparing the run-time for $1,000$ time-steps of the LBM on various discretisation levels using the existing regular grid and the newly integrated p4est grid for an increasing number of processes (left). On the right we show the relative factor that our implementation is slower.

and a constant neighbourhood. Instead, we discarded and re-created the information about the ghost layer in p4est_ghost as well as the lookup tables for the neighbouring information in p4est_mesh in each time step. The overhead of dynamical grid adaptivity was bound by a factor of 4 as shown in Table 2 and Fig. 13. We therefore could prove, that even in a worst case scenario, where the grid would be allowed to change in every fine time step, the performance of the adaptive grid does not degrade compared to an optimised regular grid. Note that this is true in spite of using a data layout that is known to reduce the performance of a regular discretisation and despite performing additional grid traversals for rebuilding p4est_ghost and p4est_mesh. We use the collision optimised data layout, because it will reduce the amount of data that has to be transferred in case of dynamic spatial adaptivity, cf. 4.3. The impact on performance for the different data layouts in case of a regular grid is described in [54].

Further optimisations reusing the overhead are to be done in the future. E.g., by restricting the grid adaptivity to the end of the coarsest time steps, we can omit rebuilding the information of p4est_ghost and p4est_mesh during the smaller time steps of the smaller cells.

5.2 Creating p4est_ghost and p4est_mesh

Building the information of neighbourhood and ghost cells in case of a regular grid can be done in constant time using simple index calculations. In case of a dynamically-adaptive grid, this process is more challenging. A comparison between creating the ghost layer and accessing neighbours in case of regular and octree-based grids is schematically shown in Fig. 14. To investigate the overhead of the more complicated ghost layer as well as the creation of lookup tables for the neighbourhood information, we performed run-time and scalability experiments using our regularly refined grids within p4est for the Poiseuille flow scenario.

We measured the run-time for creating both, p4est_mesh and p4est_ghost using a single octree with periodic boundaries for different discretisation levels and a varying number of processes. Both, ghost information and lookup tables have only been created for faces and

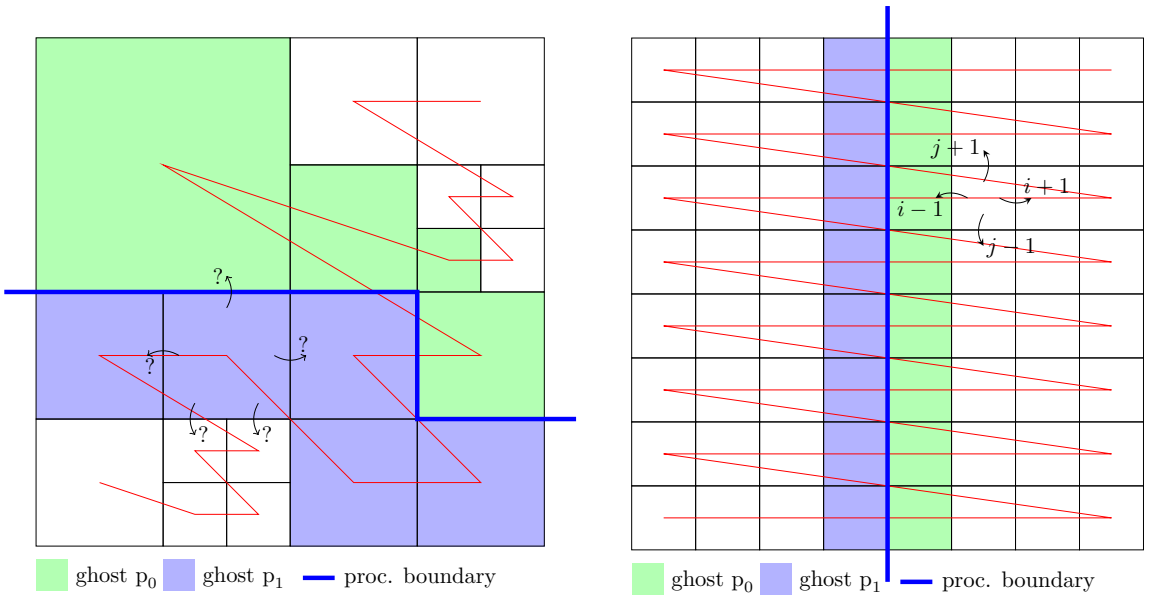

Figure 14: Differences in the structure and calculation of ghost layers and neighbour in formation in a regular grid versus a spatially adaptive grid.

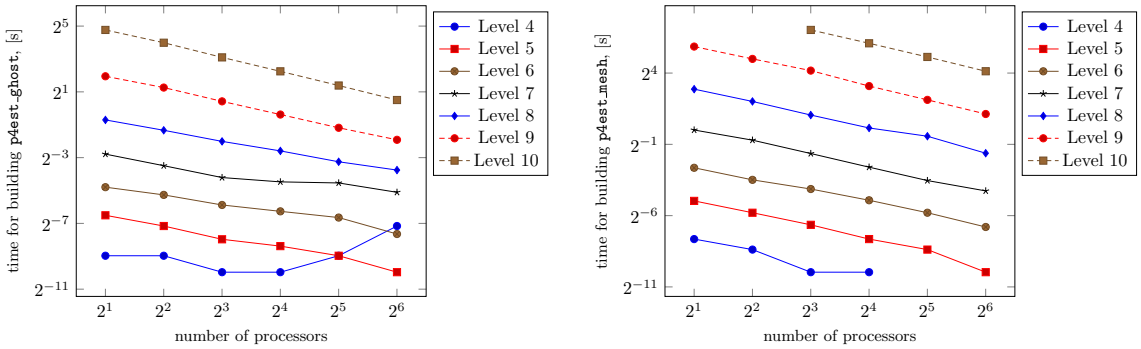

Figure 15: Left: run-times for creating `p4est_ghost`; right: run-times for creating `p4est_mesh` for a regularly refined grid on various discretisation levels for an increasing number of processes.

edges, i.e., no information about corner neighbours was gathered as this is not required by the D3Q19 lattice-Boltzmann method. The results are shown in Fig. 15. Note that the volume to surface ratio is low in the grid of level 4 which limits the scalability for creating the ghost information.

5.3 Run-Times of Grid Traversals

To compare the computational costs of traversing the grid in the streamlined and optimised way using `p4est_iterate` and the more flexible `p4est_mesh` we applied some stencil operations similar to the one used by the D3Q19 LBM. Again, we investigated the run-time for traversing a regularly discretised `p4est` based grid with periodic boundary conditions once while accessing all nearest and next-nearest neighbours. We varied both, discretisation level and number of processes. The results are shown in Fig. 16. We observe that traversing the grid using `p4est_mesh` is on average a factor of 2 slower than using `p4est_iterate`. This represents in a sense the price we pay for the minimal-invasiveness, in particular for being able to preserve the existing grid traversal logic and the direct neighbour access in ESPResSo. Note that on discretisation level four there is only a very limited number of quads associated to each

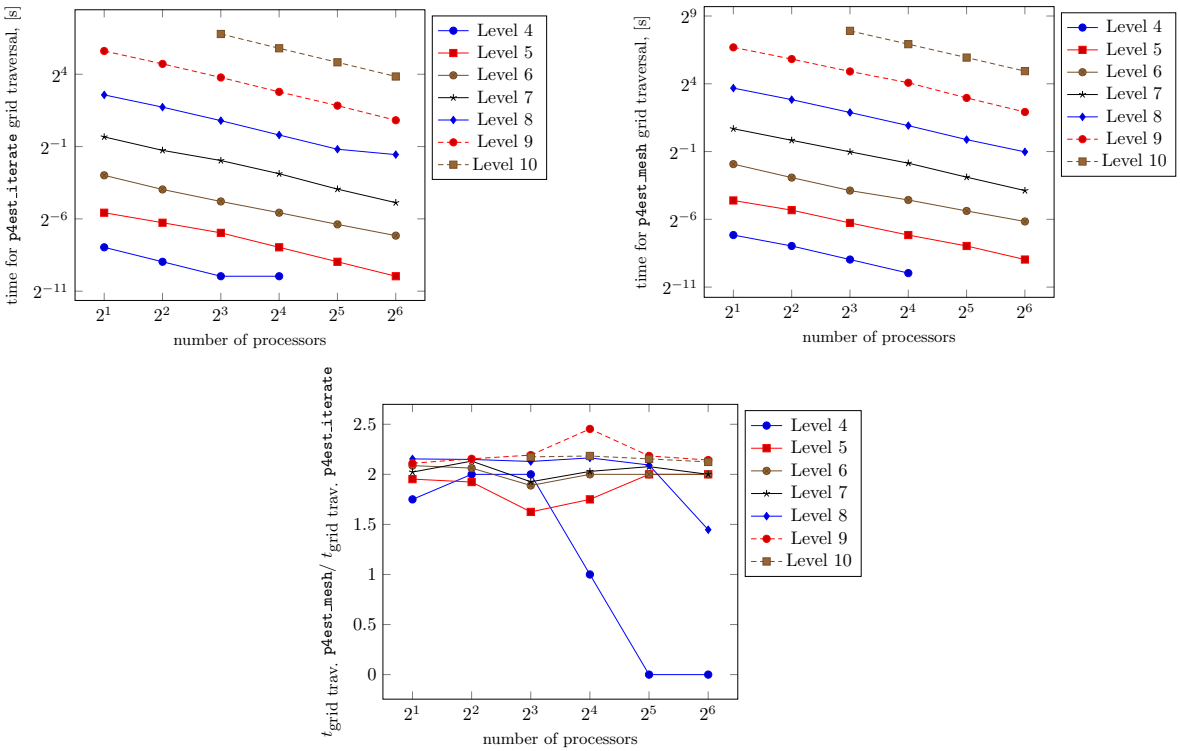

Figure 16: Comparison of the run-times of a grid traversal. Top left: using the highly optimised `p4est_iterate`; top right: using the much more flexible `p4est_mesh`; bottom: relative overhead of `p4est_mesh` compared to `p4est_iterate`. In both cases, we traverse a regularly refined grid using an increasing number of processes.

processing node such that in both cases the run-time was less than a microsecond.

6 CONCLUSION AND OUTLOOK

With the combination of ESPResSo and `p4est`, we have prepared a powerful implementation of a lattice-Boltzmann solver on adaptively refined octree grids. We are not the first group setting up such a solver. However, there are a few major differences to other approaches: [55] and [56] both implement the lattice-Boltzmann method on top of a highly optimised octree grid implementation. I.e., the solver itself had to be developed more or less from scratch in a way that is adapted to the grid structure and its specific memory-optimal traversal in the Peano framework [57] or waLBerla [58]. Also, [56] provides a solver on statically adaptive meshes, which allows for much more sophisticated solutions in terms of run-time optimisation per core and on massively parallel hardware. In contrast, we focus on full dynamical adaptivity as required by the physical systems simulated with ESPResSo, where regions with high refinement requirements move with particles or molecules immersed in the flow. In addition, the task was to port the full lattice-Boltzmann functionality in ESPResSo to the new grid type with only minimal changes to the code.

Our results show that `p4est` is well-suited for minimally-invasive integration of an octree-based grid into existing applications. Some changes in the code of the existing application are necessary, as at least the additional numerical features at boundaries between different refinement levels have to be developed from scratch and i, j, k-loops have to be exchanged by more general variants. However, most of the core code can be preserved. Even in worst-case tests where we assumed full dynamical adaptivity in every fine grid time step, we observe good performance and scalability. The flexibility and minimal invasiveness of the `p4est_mesh` comes

with an acceptable overhead when compared to specialised versions such as [58, 57].

In the future, we intend to tackle complex real world applications using the new solver. Scenarios that we expect to clearly benefit from spatial and time adaptivity are, e.g., the translocation of DNA through a nanopore [59] or the filtering of dust particles in charged cabin air filters [60]. To actually be able to handle these scenarios efficiently, the most important steps to be taken are: 1) the full implementation of the virtual cell functionalities at refinement boundaries; 2) the adaption of the iterations to level-wise iterations accounting for the natural time-adaptivity of the LBM; 3) the development of a suitable weighting concept for octree cells on different tree levels, accounting for the different time-step sizes and possible overhead at partition boundaries in the dynamical load-balancing of `p4est`; 4) the coupling of electrostatics, electrokinetics, and the molecular dynamics modules of ESPResSo to the lattice-Boltzmann solver; 5) the further optimisation of the `p4est_mesh` and its generation.

ACKNOWLEDGEMENTS

B. gratefully acknowledges travel and guest support provided by the Hausdorff Center for Mathematics funded by the DFG (German Research Foundation).

All other authors gratefully acknowledge funding provided by the collaborative research center 716 (Dynamic Simulation of Systems with Large Particle Numbers), funded by the DFG (German Research Foundation).

REFERENCES

[1] H. J. Limbach, A. Arnold, B. A. Mann, and C. Holm. ESPResSo – an extensible simulation package for research on soft matter systems. *Computer Physics Communications*, 174(9):704–727, May 2006.

[2] A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Röhm, P. Košovan, and C. Holm. ESPResSo 3.1 — molecular dynamics software for coarse-grained models. In M. Griebel and M. A. Schweitzer, editors, *Meshfree Methods for Partial Differential Equations VI*, volume 89 of *Lecture Notes in Computational Science and Engineering*, pages 1–23. Springer Berlin Heidelberg, 2013.

[3] C. Burstedde, L. C. Wilcox, and O. Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, January 2011.

[4] C. Burstedde. `p4est`: Parallel AMR on forests of octrees. `http://www.p4est.org/`, last accessed February 25, 2016.

[5] T. Weinzierl and M. Mehl. Peano—a traversal and storage scheme for octree-like adaptive Cartesian multiscale grids. *SIAM Journal on Scientific Computing*, 33(5):2732–2760, October 2011.

[6] H. Sundar, R. S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008.

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

[8] V. I. Kolobov, R. R. Arslanbekov, V. V. Aristov, A. A. Frolova, and S. A. Zabelok. Unified solver for rarefied and continuum flows with adaptive mesh and algorithm refinement. *Journal of Computational Physics*, 223(2):589–608, 2007.

[9] M. Geier, A. Greiner, and J. G. Korvink. Bubble functions for the lattice Boltzmann method and their application to grid refinement. *The European Physical Journal Special Topics*, 171(1):173–179, 2009.

[10] H. Chen. Volumetric formulation of the lattice boltzmann method for fluid dynamics: Basic concept. *Physical Review E*, 58(3):3955–3963, 1998.

[11] A. Arnold, K. Breitsprecher, F. Fahrenberger, S. Kesselheim, O. Lenz, and C. Holm. Efficient algorithms for electrostatic interactions including dielectric contrasts. *Entropy*, 15(11):4569–4588, 2013.

[12] M. Deserno and C. Holm. How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines. *Journal of Chemical Physics*, 109:7678, 1998.

[13] A. Arnold and C. Holm. A novel method for calculating electrostatic interactions in 2D periodic slab geometries. *Chemical Physics Letters*, 354:324–330, 2002.

[14] A. Arnold, J. de Joannis, and C. Holm. Electrostatics in periodic slab geometries I. *Journal of Chemical Physics*, 117:2496–2502, 2002.

[15] A. Arnold, J. de Joannis, and C. Holm. Electrostatics in periodic slab geometries II. *Journal of Chemical Physics*, 117:2503–2512, 2002.

[16] A. Arnold and C. Holm. MMM1D: A method for calculating electrostatic interactions in 1D periodic geometries. *Journal of Chemical Physics*, 123(12):144103, September 2005.

[17] C. Tyagi, M. Süzen, M. Sega, M. Barbosa, S. S. Kantorovich, and C. Holm. An iterative, fast, linear-scaling method for computing induced charges on arbitrary dielectric boundaries. *The Journal of Chemical Physics*, 132:154112, 2010.

[18] S. Tyagi, A. Arnold, and C. Holm. ICMMM2D: An accurate method to include planar dielectric interfaces via image charge summation. *Journal of Chemical Physics*, 127:154723, 2007.

[19] S. Tyagi, A. Arnold, and C. Holm. Electrostatic layer correction with image charges: A linear scaling method to treat slab 2D + h systems with dielectric interfaces. *Journal of Chemical Physics*, 129(20):204102, 2008.

[20] J. J. Cerdà, V. Ballenegger, O. Lenz, and C. Holm. P3M algorithm for dipolar interactions. *Journal of Chemical Physics*, 129:234104, 2008.

[21] D. Roehm and A. Arnold. Lattice Boltzmann simulations on GPUs with ESPResSo. *The European Physical Journal Special Topics*, 210:89–100, 2012.

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

[23] J. Höpfner, T. Richter, P. Košovan, C. Holm, and M. Wilhelm. Seawater desalination via hydrogels: Coarse grained simulations and practical realisation. *Progr. Colloid. Polym. Sci.*, 140(140), 2013.

[24] B. J. Reynwar, G. Illya, V. A. Harmandaris, M. M. Muller, K. Kremer, and M. Deserno. Aggregation and vesiculation of membrane proteins by curvature-mediated interactions. *Nature*, 447(7143):461–464, May 2007.

[25] M. Kuron and A. Arnold. Role of geometrical shape in like-charge attraction of dna. *European Physical Journal E: Soft Matter*, 38:20, 2015.

[26] S. Kesselheim, M. Sega, and C. Holm. Effects of dielectric mismatch and chain flexibility on the translocation barriers of charged macromolecules through solid state nanopores. *Soft Matter*, 8(36):9480–9486, 2012.

[27] K. Breitsprecher, P. Košovan, and C. Holm. Coarse grained simulations of an ionic liquid-based capacitor i: density, ion size, and valency effects. *Journal of Physics: Condensed Matter*, 26:284108, 2014.

[28] S. Succi. *The lattice Boltzmann equation*. Oxford university press New York, 2001.

[29] B. Dünweg and A. J. C. Ladd. Lattice boltzmann simulations of soft matter systems. In *Advanced Computer Simulation Approaches for Soft Matter Sciences III*, volume 221 of *Advances in Polymer Science*, pages 89–166. Springer-Verlag Berlin, Berlin, Germany, 2009.

[30] S. Succi, M. Sbragaglia, and S. Ubertini. Lattice boltzmann method. *Scholarpedia*, 5(5):9507, 2010.

[31] K. Grass. *Towards realistic modelling of free-solution electrophoresis: a case study on charged macromolecules*. PhD thesis, Goethe-Universität Frankfurt, 2009.

[32] L.-S. Luo, W. Liao, X. Chen, Y. Peng, and W. Zhang. Numerics of the lattice boltzmann method: Effects of collision models on the lattice boltzmann simulations. *Physical Review E*, 83(5):056710, 2011.

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

[34] B. Dünweg, U. Schiller, and A. J. C. Ladd. Statistical mechanics of the fluctuating lattice-boltzmann equation. *Phys. Rev. E*, 76:36704, 2007.

[35] W. C. Rheinboldt and C. K. Mesztenyi. On a data structure for adaptive finite element mesh refinements. *ACM Transactions on Mathematical Software*, 6(2):166–187, 1980.

[36] D. A. Kopriva, S. L. Woodruff, and M. Y. Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method. *International Journal for Numerical Methods in Engineering*, 53(1):105–122, 2002.

[37] D. Rosenberg, A. Fournier, P. Fischer, and A. Pouquet. Geophysical-astrophysical spectral-element adaptive refinement (GASpAR): Object-oriented h-adaptive fluid dynamics simulation. *Journal of Computational Physics*, 215(1):59–80, 2006.

[38] M. Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Texts in Computational Science and Engineering. Springer, 2012.

[39] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[40] G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM Ltd., 1966.

[41] M. Griebel and G. W. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. *Parallel Computing*, 25:827–843, 1999.

[42] H.-J. Bungartz, M. Mehl, and T. Weinzierl. A parallel adaptive Cartesian PDE solver using space–filling curves. *Euro-Par 2006 Parallel Processing*, pages 1064–1074, 2006.

[43] T. Tu, D. R. O'Hallaron, and O. Ghattas. Scalable parallel octree meshing for terascale applications. In *SC '05: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE, 2005.

[44] C. Burstedde, O. Ghattas, M. Gurnis, E. Tan, T. Tu, G. Stadler, L. C. Wilcox, and S. Zhong. Scalable adaptive mantle convection simulation on petascale supercomputers. In *SC08: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM/IEEE, 2008.

[45] R. S. Sampath and G. Biros. A parallel geometric multigrid method for finite elements on octree meshes. *SIAM Journal on Scientific Computing*, 32(3):1361–1392, 2010.

[46] J. R. Stewart and H. C. Edwards. A framework approach for developing parallel adaptive multiphysics applications. *Finite Elements in Analysis and Design*, 40(12):1599–1617, 2004.

[47] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24, 2007.

[48] T. Isaac, C. Burstedde, and O. Ghattas. Low-cost parallel algorithms for 2:1 octree balance. In *Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2012. `http://dx.doi.org/10.1109/IPDPS.2012.47`.

[49] M. Mirzadeh, A. Guittet, C. Burstedde, and F. Gibou. Parallel level-set methods on adaptive tree-based grids. Submitted, 2015.

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

[51] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas. An extreme-scale implicit solver for complex pdes: highly heterogeneous flow in earth's mantle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 5. ACM, 2015.

[52] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo. Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA. http://arxiv.org/abs/1511.01561, 2015.

[53] M. Rohde, D. Kandhai, J. J. Derksen, and H. E. A. van den Akker. A generic, mass conservative local grid refinement technique for lattice-boltzmann schemes. *International Journal for Numerical Methods in Fluids*, 51(4):439–468, 2006.

[54] G. Wellein, T. Zeiser, G. Hager, and S. Donath. On the single processor performance of simple lattice Boltzmann kernels. *Computers and Fluids*, 35(8-9):910–919, September 2006.

[55] P. Neumann and T. Neckel. A dynamic mesh refinement technique for lattice boltzmann simulations on octree-like grids. *Computational Mechanics*, 51(2):237–253, January 2013.

[56] C. Feichtinger, J. Habich, H. Köstler, G. Hager, U. Rüde, and G. Wellein. A flexible patch-based lattice boltzmann parallelization approach for heterogeneous gpu-cpu clusters. *Parallel Computing*, 37:536–549, 2011.

[57] T. Weinzierl and M. Mehl. Peano — a traversal and storage scheme for octree-like adaptive cartesian multiscale grids. *SIAM Journal on Scientific Computing*, 33(5):2732–2760, January 2011.

[58] C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüde. WaLBerla: HPC software design for computational engineering simulations. *Journal of Computational Science*, 2:105–112, 2011.

[59] S. Kesselheim, M. Sega, and C. Holm. Applying ICC$\star$ to dna translocation: Effect of dielectric boundaries. *Computer Physics Communications*, 182(1):33–35, 2011.

[60] M. J. Lehmann, J. Weber, A. Kilian, M. Heim, and MANN+HUMMEL GmbH, Germany. Micro scale simulation as part of fibrous filter media development processes - from real to virtual media. Feb 2015.