Eur. Phys. J. Special Topics **227**, 1757–1778 (2019)

https://doi.org/10.1140/epjst/e2019-800161-5

THE EUROPEAN PHYSICAL JOURNAL SPECIAL TOPICS

Review

Adaptive grid implementation for parallel continuum mechanics methods in particle simulations

Miriam Mehl[a] and Michael Lahnert

Institute for Parallel and Distributed Systems, Universitätsstr. 38, 70569 Stuttgart, Germany

Received 1 October 2018 / Received in final form 1 December 2018
Published online 8 March 2019

Abstract. In this tutorial review paper, we present our minimally invasive approach for integrating dynamically adaptive tree-structured grids into existing simulation software that has been developed for regular Cartesian grids. We introduce different physical models that we target and that span a wide range of typical simulation characteristics – from grid-based Lattice-Boltzmann, finite volume and finite difference discretized models to particle-based molecular dynamics models. We derive the respective typical data access requirements and extensions of the algorithms to adaptively refined grids along with possible grid adaptivity criteria. In addition, after introducing basics of tree-structured adaptively refined grids, we present the adaptive grid framework `p4est` and our enhancement of `p4est` in order to provide a grid and partitioning infrastructure that can easily be used in existing simulation codes. Finally, we explain how such a grid infrastructure can be integrated into regular grid codes in general in three major steps and how we integrated `p4est` in the soft matter simulation package ESPResSo in particular. A summary of results from previously published performance and scalability studies together with new results for more realistic coupled simulation scenarios shows the efficiency and validity of the resulting new version of ESPResSo.

1 Introduction

1.1 Motivation

Adaptive mesh refinement (AMR) is a highly efficient tool to extend domain sizes and time spans that can be tackled in a simulation. Such grids allow covering large volumes and time intervals while keeping the number of data points and, therewith, operations at a tolerable level. If the simulated scenario changes over time, the distribution of local areas of high grid resolutions in the simulation domain has to be adapted over time. Tree-structured adaptive grids such as octrees feature a clearly defined refinement structure leading to substantially lower memory requirements compared to unstructured grids. At the same time, they offer efficient load-balanced

[a] e-mail: miriam.mehl@ipvs.uni-stuttgart.de

partitioning capabilities and arbitrarily local and dynamic refinement options. In literature, many approaches to implement adaptive tree-structured grids have been presented. Usually, the details of the grid implementation are encapsulated within a library. Grid libraries range from highly optimized approaches in terms of memory usage that restrict the order for storage and cell access to tree-traversals along specific space-filling curves (and, thus, do not allow direct neighbor cell access) [1] to more generic concepts such as those in [2–4] that are based on the Morton curve. In [5] the Hilbert curve is used for partitioning between different ranks while within ranks refinement patches are used for locally refining the grid. More than ten years ago, codes like Dendro [2] or Octor [6] proved that tree-structured grids can scale to several ten-thousand cores. This was surpassed by codes like waLBerla [7,8] or `p4est` [3] that scales to hundred-thousand cores. Moreover, task-based parallelism has been realized e.g., Uintah [9] and recently, AMR algorithms have been implemented on GPUs [10]. A comparison of more libraries for tree-structured as well as block-structured grids has been published in [11].

The other side of the coin is that existing large simulation packages have often been developed for regular grids only, and features that are trivial in regular grids such as direct neighbor access via (i, j, k)-indices for grid cells are heavily used throughout the whole code basis. The aim of this paper is to present ways to use tree-structured adaptive grids in such codes in a minimally invasive way and still exploit their efficiency potential as far as possible. From the efficiency perspective, every simulation algorithm should follow the tree-structure (in grid traversals and data storage order). This, however, drastically changes the data access pattern and, therewith, requires a major (infeasible) re-factoring of existing regular grid codes. As a consequence, many applications represent tree-structured grids in the same way as unstructured grids, i.e., storing all connections between grid nodes, edges, faces, and volumes. Our approach is to avoid the resulting loss of efficiency by using a sophisticated dedicated adaptive grid framework with efficient interfaces for neighbor cell access.

As today's and even more future computer architectures require highly efficient algorithms in terms of memory usage and memory access in combination with an immense level of parallelism, a particular focus has to be on efficient data-structures that still allow using standard access patterns in existing simulation codes. Data have to be accessible based on simple iterators, i.e., loops over grid elements. To make optimal use of cache-hierarchies, the underlying data structures should in addition provide high spatial and temporal locality for data access and maintain this property also after dynamic refinement or coarsening. We demonstrate this approach for the example of the soft matter simulation package ESPResSo [12,13] and base our grid implementation on the re-known grid framework `p4est` [3]. Note that, following the idea of minimally invasive integration, we do not alter existing modeling and algorithmic choices in ESPResSo except for enhancements required to allow for dynamical grid adaptivity. Our aim is neither to present new physical modeling aspects nor to achieve the highest possible efficiency for single physical component. Instead, we provide a feasible way to enhance a simulation software that includes decades of experience and knowledge carefully such that the whole functionality of all components and their coupling is maintained and a good efficiency and parallel scalability is achieved.

1.2 Tree-structured adaptive grids in a nutshell

Tree-structured grids are usually Cartesian grids[1]. Thus, they can be interpreted as a straightforward generalization of regular Cartesian grids. Their construction principle

[1] Tree-structured grids based on triangular elements are used, e.g., [14].

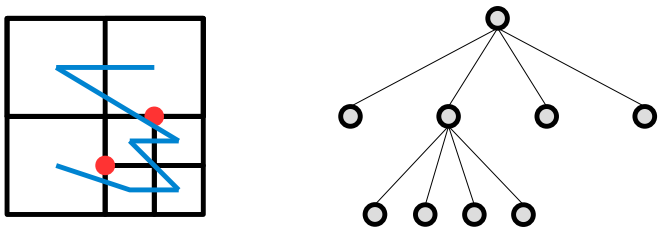

Fig. 1. Simple example for a two-dimensional quadtree grid (left) with the corresponding tree (right) and the discrete iterate of the Morton curve that can be used to linearize the tree (blue). Hanging corners of the adaptive grid are marked in red.

is based on recursive refinement of grid cells of fixed shape into a fixed number of children with equal shape. Thus, each grid corresponds to a tree as displayed in Figure 1 for a simple two-dimensional quadtree grid. Though the shape of the grid cells is the same as in regular grids, the following fundamental differences require numerical and technical enhancements if a regular grid simulation is to be enhanced to adaptive grids:

(1) Transitions between different refinement regions require modified discretization approaches. The main difficulty is the handling of hanging entities, i.e., corners (and edges in 3D) of finer grid cells that are not part of a corner (or edge) of a coarser neighbor cell. Hanging corners are marked in red in Figure 1 to illustrate this.
(2) Data stored on the grid such as physical variables need to be stored and accessed in a defined order. Due to the non-regular refinement, the respective grid cell ordering has to be different from the usual lexicographic (i, j, k)-indexing used for regular grids. Space-filing curves [15] are a widely used approach to tackle this challenge.
(3) The neighborhood of grid cells is less regular in adaptive grids, which not only requires new discretization methods as mentioned above, but also new functions that allow accessing arbitrary neighbors from each grid cell. To avoid recursions with undefined depth in the respective neighbor search, we restrict our adaptive grids to so-called 2:1 balanced grids. In such a grid, the level difference between a grid cell and all its neighbors is bounded to at most one.
(4) Due to the inhomogeneous refinement structure of the adaptive grid, simple rectangular domain partitioning does no longer lead to load-balanced parallelization. Partitioning the linearized list of grid cells provided by a space-filling curve ordering into equal pieces is a simple and quasi-optimal solution for this aspect [2–4].

We discuss our solutions to all these aspects for the physical model components of ESPResSo throughout this paper.

1.3 Contributions

In this paper, we present the integration of the grid framework `p4est` with ESPResSo. To this end, we

(1) develop requirements for direct data access derived from various simulation components in ESPResSo comprising a Lattice-Boltzmann method, a finite volume discretization of electrokinetic equations, a finite difference discretization for long-range electrostatics, and various short-range particle interactions; this includes the derivat ion of generalizations of all these components to adaptive grids;

(2) present extensions of `p4est` that have been developed and implemented to fulfill these requirements;
(3) integrate the respective grids in the mentioned components of ESPResSo, including the exchange of (i, j, k)-based neighbor access, mesh partitioning and ghost layer communication by `p4est` functions; in addition, we describe the dynamic refinement/coarsening implementation;
(4) showcase physical results together with a performance and scalability analysis in the results section.

The remainder of the paper is structured as follows: We present the physical model components in ESPResSo that we target for simulation on adaptive grids in Section 2. Section 3 introduces the grid framework `p4est` and presents the enhancements that were required for a grid implementation that can be integrated in ESPResSo in a minimally invasive and efficient way. The integration of the adaptive grids in ESPResSo itself is presented in Section 4. Finally, we present a summary of previous efficiency and scalability studies as well as results for two new real-world scenarios in Section 5.

2 Modeling and discretization in space and time on adaptive grids

In this section, we give an overview of the target application, i.e., various model components and their solvers as implemented in the soft matter simulation package ESPResSo. We give a very short overview of the software package ESPResSo, describe the model components and their discretization and algorithmic choices as used in ESPResSo in the regular grid version. In addition, we derive required extensions for dynamically adaptive grids.

2.1 ESPResSo in a nutshell

ESPResSo [12,13] is a large-scale extensible simulation package for coarse grained soft matter simulation. It implements electrostatics, electrokinetics, short-range molecular dynamics, and hydrodynamics (Lattice-Boltzmann), both for periodic and non-periodic systems. All models can be flexibly combined with each other yielding complex coupled simulations. Applications range from engineering problems such as soot aggregation [16], hydrogels [17], biological membranes [18], DNA like-charge attraction [19] and translocation [20], to ionic liquids [21,22].

ESPResSo is a free, open-source software published under the GNU General Public License (GPL3). To achieve computational efficiency, ESPResSo has been implemented in C and C++. A Python scripting interface provides an easy-to-use way to configure arbitrarily complex coupled simulations and modify parameters. The parallelization is based on MPI. GPU acceleration can be employed for Lattice-Boltzmann, electrostatics, and electrokinetics calculations using a single GPU. In the version before this work, ESPResSo was based on regular Cartesian grids and Linked-Cell systems. For many applications, the simple regular spatial discretization prevented simulating larger time and length scales that are, however, required to achieve new insights in the analyzed physical systems.

2.2 Lattice-Boltzmann

The Lattice-Boltzmann method (LBM) [23–25] is a well-established alternative to the numerical solution of the Navier–Stokes equations. The Lattice-Boltzmann equation

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

is derived from the Boltzmann equation via discretization of the velocity space with a chosen set of distinct velocities $\mathbf{c}_i$ (see Fig. 2 (left)). f_i denote probability densities (called moments or populations), Ψ_i external forces.

In each LBM time step, two main steps are executed: (1) a collision step defined by the operator L_{ik} relaxing cell-local probabilities towards the equilibrium f_k^{eq} and (2) the streaming step transporting densities to neighbor cells according to their flow direction.

There are multiple relaxation-type models for the collision step with constant [26–30] or variable [31,32] relaxation rates. In these schemes, the populations are relaxed towards a local equilibrium derived from the Maxwell–Boltzmann distribution. These classical schemes have certain short-comings, especially when it comes to turbulent flows which can be mitigated by more elaborate collision models: Systematically deriving a higher order equilibrium function [33] leads to a more stable scheme at larger velocities [34]. Galilean invariance is not retained if physical moments are relaxed at different rates within a static frame. In cascaded LBM [35,36], distributions are transformed to so-called central moments. This fixes instabilities at very low viscosities (*zero viscosity limit*). Cumulant LBM [37,38] increases the stability of the LBM algorithm further by relaxing cumulants instead of moments. This solves any issues with Galilean invariance, which is particularly relevant in turbulent flow scenarios [39,40]. In terms of important properties for our grid infrastructure, the algorithmic characteristic is always that of a cell-local interaction between densities f_i of a single cell, in spite of different physical properties.

The streaming step propagates the result of the collision step to the respective neighbor cells. We use a double buffering scheme to realize this step, i.e., first propagate densities f_i to auxiliary variables $\tilde{f}_i$ of the neighbor cells and, in a second grid traversal, swap $\tilde{f}_i$ and f_i in each grid cell[2]. Boundary conditions at outer walls and for complex obstacles are realized based on a first order accurate bounce-back scheme in ESPResSo [43]. More accurate boundary conditions, e.g. for curved boundaries or partial slip, are presented in [44,45].

2.2.1 Lattice-Boltzmann methods on adaptive grids

In the Lattice-Boltzmann method on regular grids, the time-step Δt is chosen such that particle densities reach the next neighbor in the direction of their velocity $\mathbf{c}_i$ in one time step. One way to generalize the algorithm to adaptive grids is to use the same time step but a different Courant–Friedrichs–Lewy (CFL) number on all grid levels [46,47]. Another way is to keep the relation between mesh width and time step, thus introducing space-time adaptivity (so-called acoustic scaling). There are two main models using multi-variate time-stepping: Temporal interpolation techniques [48–50] impose boundary conditions in an overlap region and require scaling the non-equilibrium part of the distributions. *Compact interpolation* allows obtaining a small overlap region and interpolation of quadratic order using second-order bubble functions [51]. Volumetric formulations [52–54] use an overlap region of the size of

[2] This yields a simple way to avoid overwriting of data independent of particular properties of the grid traversal. More sophisticated streaming schemes have been published that avoid double buffering, e.g., [41,42]. We stick to the double buffering scheme following our idea of minimally-invasive integration and consistency with regular grid and GPU code versions of ESPResSo that both use the double-buffering scheme.

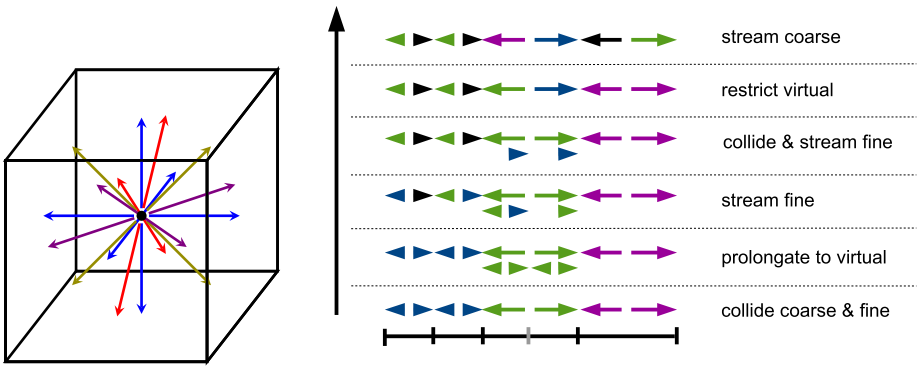

Fig. 2. Left: The D3Q19 stencil consists of a resting velocity (black), six velocities parallel to the coordinate axes (blue) and twelve velocities pointing to the twelve edges of the cell (red, olive, and violet). Right: algorithmic steps of the LBM time stepping scheme for a simple two-level 1D adaptive grid. Reading from bottom up: (i) collision in real cells on both levels; (ii) interpolation to virtual fine cells; (iii) streaming in fine real and virtual cells; (iv) collision (in real cells only) and streaming (in real and virtual cells) on the fine level; (v) restriction to coarse real cells; (vi) streaming in coarse cells. This scheme is recursively expanded to larger level differences.

the coarse cell at the refinement boundary to conserve mass, momentum, and viscous stresses [55][3].

We use the volumetric scheme proposed by Rohde et al. [54]. Accordingly, we have to define an execution order of collision and streaming with different time steps on different levels and modify the streaming step when cells have direct neighbors of different size. The scheme proposes adding virtual children within each real grid cell with finer neighbor cells[4]. We adopt this idea and, thus, in the streaming step, data are only exchanged between cells, real or virtual, of the same refinement level. The respective steps of a coarse grid time step at a refinement boundary are illustrated for a 1D case in Figure 2 (right).

Refinement or coarsening of the adaptive grid can be triggered based on the evaluation of geometric criteria (close to boundaries or around submerged particles) or physical properties such as flow velocities or vorticity.

Neighbor access requirements. An important conclusion drawn from the algorithm described above are neighbor access requirements of grid cells as a basis for the streaming step. We have described these requirements for real and virtual cells in detail in [58]. In this paper, we summarize the most important results: Streaming between real cells of a given level requires detecting all neighbors that are defined as relevant by the used stencil. In addition, streaming is required in all directions of the stencil also between real cells and virtual neighbors. For the streaming between virtual cell, more complex cases have to be distinguished. Basically, streaming is not required if the respective densities don't move towards or come from neighboring fine grids cells. As this causes complex cases in the respective code, we propose an alternative approach in [58] that detects neighbor relations between virtual cells at the same refinement level in all directions and also executes the respective streaming step. One can show that, due to our particular time stepping algorithm and the double

[3] When changing the (local) grid resolution, further variables have to be re-scaled accordingly. This re-scaling, however, does not affect the requirements for our grid infrastructure, see Rohde et al. [54] and Lahnert et al. [56] for details.

[4] This has been implemented, e.g., PEANO [57] or waLBerla [7,8].

buffering approach, the respective steaming steps that would not have been strictly required are overwritten by the subsequent coarse grid streaming. At outer domain boundaries, we can show that also the implementation of bounce-back boundary conditions is correct if implemented analogously.

A different situation occurs at boundaries between two partitions in the distributed memory parallelization: Here, a process only needs to detect virtual cells in the real cells of its ghost layer if they are required due to finer cell in its own partition. This insight substantially reduces communication requirements between partitions in the generation step for ghost layers including virtual cells.

2.3 Electrokinetics

The electrokinetic model which was first introduced in [59] is a continuous representation for ion transport in a low Reynolds number flow. We base our notation on [60]. It consists of three main components: (1) ionic flux, (2) electrostatic interactions, and (3) hydrodynamics. In this section, we focus only on the ionic flux calculation. The flow is calculated with the Lattice-Boltzmann method described above, whereas the solver for electrostatic interactions is presented in the following subsection. For two ionic species, a positive ionic species with density c_+ and ionic flux $\mathbf{j}_+$ and a negative species with density c_- and ionic flux $\mathbf{j}_-$, the ionic flux equations read:

$$\partial_t c_\pm = -\nabla \cdot \mathbf{j}_\pm \tag{2}$$

$$\mathbf{j}_\pm = \underbrace{-D_\pm \nabla c_\pm - \mu_\pm z_\pm e c_\pm \nabla \Phi}_{\mathbf{j}_\pm^{\text{diff}}} + \underbrace{c_\pm \mathbf{u}}_{\mathbf{j}_\pm^{\text{adv}}} \tag{3}$$

where (2) is the continuity equation for the species density fields $c_\pm$ and their ionic flux $\mathbf{j}_\pm$. Equation (3) is a diffusion-advection equation for the total ionic flux, e denotes the elementary charge, $z_\pm$ the species valencies, $\mu_\pm$ the species mobility, $D_\pm$ the diffusion coefficients, Φ the electrostatic potential (calculated in the electrostatics solver), and $\mathbf{u}$ the fluid field (calculated in the Lattice-Boltzmann flow solver)[5].

The electrokinetic equations are solved in ESPResSo based on a finite volume discretization and explicit time stepping. As fluxes are propagated according to the finite volume representation of the continuity equation, we ensure density conservation down to arithmetic machine precision. To cope with different refinement levels, virtual cells are embedded at refinement boundaries. As physical criteria for grid adaptivity, gradients of the densities or of the potential can be used.

2.3.1 Neighbor access requirements

Diffusion is calculated on a D3Q18 lattice while advection uses a D3Q27 lattice. Therefore, diffusion uses the same stencil as the LBM (albeit diffusion does not have a (0, 0, 0) vector) while advection also needs neighbor access across corners.

2.4 Electrostatics

The model for electrostatics is based on the Coulomb potential, i.e., described by long-range molecular dynamics. In the original implementation on regular Cartesian

[5] In non-equilibrium conditions, the model needs to be extended by additional force density terms as proposed by Rempfer et al. to avoid spurious flow. We do not detail this aspect in the paper, but refer to literature.

grids, the respective potential field Φ is calculated by Ewald summation [61,62], the forces on particles are computed by numerical differentiation. This implementation has been improved in [64] to a grid-based version that allows approximating Φ with efficient discrete fast Fourier transformation (P^3M, [65,66]). Using FFT, however, enforces setting globally homogeneous permittivity. The multitude of implemented algorithms in ESPResSo has been compared in [67].

To generalize this approach to adaptive grids, the FFT-based potential solver has been substituted by a successive over-relaxation (SOR) scheme for the corresponding Poisson equation

$$\nabla \cdot (\varepsilon_s \nabla \Phi) = \sum_{\pm} z_{\pm} e c_{\pm} \tag{4}$$

where ε_s describes the permittivity. To avoid interpolation errors, equation (4) is discretized on the same grid as the finite volume scheme for electrokinetics.

2.4.1 Neighbor access requirements

Equation (4) is discretized with a seven-point finite difference stencil. Analogous to LBM and electrokinetics, different levels of refinement are treated by embedding virtual cells in coarse quadrants at refinement boundaries. Thus, there are no additional requirements regarding neighbor access compared to LBM or electrokinetics.

2.5 Short-range molecular dynamics

As a standard example for short-range molecular dynamics, we consider the Lennard-Jones potential [68,69]

$$V_{LJ} = 4\varepsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right), \tag{5}$$

which consists of two major physical aspects: Pauli repulsion, modeled by $\left(\frac{\sigma}{r}\right)^{12}$ and van der Waals attraction, modeled by $\left(\frac{\sigma}{r}\right)^{6}$. ε describes the minimum of the potential and σ models the distance where the potential vanishes. Due to the fast decay of the potential, the influence of the interaction is neglected at a certain distance, the so-called cut-off radius r_c. The Linked-Cell method provides an efficient implementation of this cut-off approach. The domain is discretized with a regular Cartesian grid, in the classical methods with grid spacing r_c, into which the molecules are inserted according to their position[6]. This guarantees that all interaction partners of each molecule can be found in the adjoining cells. As we have not ported the Linked-Cell algorithm itself but provide a way to populate ESPResSo's internal interaction lists, the discretization remains regular.

2.5.1 Neighbor access requirements

Similar to the electrokinetics algorithm, direct access to neighbor cells is required across all entities (faces, edges, and corners). Here, however, only regular cells have to be considered.

[6] Adaptive Linked Cell grids have been shown to not be superior to regular Linked cells with mesh width r_c, see for example [70].

2.6 Coupling

To integrate all the above described physical components with each other ESPResSo offers a bidirectional force-coupling. Grid-based algorithms (LBM, electrokinetics, electrostatics) may choose a larger time step than the particle simulation. However, as all time steps are explicit, no interpolation in time is required. All time steps in all components have a fixed size that is chosen a-priori.

If all physical components are involved the ordering within a timestep looks as follows: (1) Short-range MD, (2) diffusive flux (EK), (3) advective flux (EK), (4) electrostatic potential (ES), and (5) fluid (LBM). All these components interact by a frictional, bi-directional force-coupling.

$$\mathbf{f}_{pot,fl} = \varrho \mathbf{E} = -\sum_{\pm} z_{\pm} e c_{\pm} \nabla \Phi \tag{6}$$

$$\mathbf{f}_{fl,i} = -\zeta \left(\mathbf{v}_i - \mathbf{u}(\mathbf{x}_i, t) \right) + \mathbf{f}_{st,i}. \tag{7}$$

Equation (6) describes the force that is applied on the fluid by a charge density ϱ and electric field $E = -\nabla\Phi$. The fluid force acting on a particle is given by equation (7). The difference of the particle's velocity $\mathbf{v}_i$ and the interpolated fluid velocity at the particle's position $\mathbf{u}$ yields a force which is scaled by the friction coefficient ζ. In addition, a stochastic force-term $f_{st,i}$ is added to account for Brownian motion.

3 Extensions of the grid framework `p4est`

3.1 `p4est` in a nutshell

For the work presented in this paper, we use the adaptive grid framework `p4est` [3] as a basis. `p4est` is free and open-source software (licensed under the GNU Public License version 2), written in C, and provides data structures and scalable parallel algorithms for adaptive mesh-refinement. `p4est` allows performing simulations based on grids that are composed of several octrees – so-called forests of octrees. The root of each octree is a hypercube and every cell has a unique index in the tree. The composition of several octrees allows handling non-cubic domains. `p4est` can enforce 2:1 balancing within each tree as well as across tree boundaries, such that two adjacent cells are never more than one level apart. Dynamic grid adaptivity (refinement and coarsening) is executed based on user-defined decisions for each cell. While `p4est` is capable of recursive refinement/coarsening, we restrict ourselves to changing the size of a cell by at most one level when we perform an adaptive step. After adapting the grid, we have to re-establish 2:1 balance by having `p4est` automatically refine the respective cells.

`p4est` stores octrees in a linear leaf-only structure, i.e., storing of parent tree-nodes is omitted. All leafs are stored in Morton order. Simulation data can either be stored as a struct inside the `p4est` data-structure or in separate user defined data structures. In the latter case, the numerical payload is transparent to `p4est`. That means the user is responsible for data to be allocated, moved, transferred, freed, ... after each grid operation. `p4est` provides helper functions for these tasks, that, however, have to be called manually. For parallelization, `p4est` uses Morton order based domain partitioning. The provided partitioning functionality includes the automatic generation of ghost layers (one layer of adjacent cells around each partition) and MPI communication routines between partitions.

An important aspect for the work presented in this paper is the access of neighbor data. Per se, the linear leaf-only storage does not provide direct access to neighbors of

cells. If memory is a limiting factor, neighbors will be constructed by first computing their tentative Morton index and performing a binary search for this index in all cells of the respective partition (including ghost cells). In `p4est`, this search has been optimized by traversing all interfaces between face-, edge-, and corner-adjacent cells exactly once in `p4est_iterate` [71]. If memory is not extremely limited, $\mathcal{O}(1)$ lookups of cells are, however, the fastest option. Respective lookup tables can be generated in `p4est` with `p4est_mesh` for 2:1 balanced trees. The tables contain indices, relative size, orientation, and owner process of all neighbors of a partition's local cells. In the work presented here, the lookup data have been completed to include neighbor relations between all face, edge, and corner neighbors for real and virtual cells in 2:1 balanced meshes [56,72]. We give more details below.

3.2 New components and functions

To leave application algorithms (almost) untouched, we have to provide random-access to any neighboring cells. Accordingly, we completed the neighbor lookup in `p4est_mesh` for three-dimensional grids to also provide direct access to edge and corner neighbors. To cope with the challenges from spatial adaptivity, in particular hanging nodes, virtual cell have to be introduced as described in Section 2. We realized this by `p4est_virtual`, a light-weight extension for `p4est_mesh`. We keep virtual cells transparent to `p4est`, i.e., we do not store them in the tree as real quadrants. Instead, we just tag any quadrant that has virtual children. This has the advantages that 1. by design all virtual quadrants have to reside on the same process as their host and 2. dynamic grid adaptivity is not affected by virtual cells[7].

Summing up, `p4est_virtual` performs three important tasks: (1) It marks all coarse cells at refinement boundaries hosting virtual quadrants and allows mapping user-managed (virtual cell) payload to real cells' ids. (2) It includes virtual cells in the ghost exchange and stores the respective data in a separate data-structure called `p4est_virtual_ghost`. As described in Section 2, the decision to generate virtual children for real ghost layer cells is taken locally without additional communication. (3) It offers a neighbor-lookup function to find virtual cells as neighbors and to find neighbors of virtual cells by an extension of the neighbor-lookup provided by `p4est_mesh`.

To abstract the additional complexity of traversing specific regions of the grid (like quadrants adjacent to the process boundary or quadrants of a specific level) and random-access, we implemented an iterator based on `p4est_mesh` and `p4est_virtual`.

4 Minimally invasive integration of `p4est` grids in ESPResSo

4.1 Integration of adaptive `p4est` grids

Adaptive `p4est` grids have been integrated in all four components of ESPResSo described in Section 2: Lattice-Boltzmann, electrostatics, electrokinetics, and short-range molecular dynamics, partly in cooperation with other groups. In general, to integrate `p4est` in a minimally invasive way into existing applications, we propose a three-step process [56].

[7] In particular, this is important for coarsening, where children of the same father cell can only be coarsened if all cells are local quadrants on the same MPI rank. This property can be enforced for one level above leaf level only. Thus, we have to enforce it for the hosts of virtual cells.

(1) First, the regular grid needs to be substituted by a `p4est` based grid. It allows performing simulations on regularly discretized `p4est` grids which may already contain dynamic load balancing if there are different loads for each cell. This first step comprises the following substeps:
   (a) The (i, j, k) loops over regular grids have to be replaced by `p4est` iterators. Neighbor access changes from simply incrementing the regular grid indices i, j, or k to lookups using `p4est_mesh`. The physical kernel functionalities such as LBM collision, streaming, stencil evaluations, or particle interactions, however, remain unchanged.
   (b) Morton curve-based partitioning provided by `p4est` substitutes the original (rectangular) domain partitioning in ESPResSo. This comprises the automatic generation of ghost layers by `p4est` and replacing all ESPResSo inter-partition communication steps by calls of `p4est` communication routines.

(2) Second, the algorithms are extended to actually use an adaptive grid. Additional functions have been added for new numerical steps required at refinement boundaries (in particular interpolation and restriction to and from virtual cells[8]). As a basis for all these steps, data structures for virtual cell payload have to be provided. Dynamic grid adaptivity is triggered by new adaptivity criteria that have to be implemented in the respective components. The generation of the adapted grid follows the ideas in [73]. We generate a new `p4est`-instance which we refine, coarsen, and balance. Subsequently, we map the user data (payload) from the old to the new grid. In refinement or coarsening regions, this requires interpolation or restriction, respectively. If virtual cells are needed, their generation requires additional grid traversals.

(3) Third, the implementation of the application needs to be optimized, e.g., by implementing communication hiding or by modifying data-access patterns.

In the following paragraphs, we briefly describe the component-specific changes and enhancements of simulation modules in ESPResSo that go beyond or specify the steps of this three-step integration. More details are given in our paper [58] focusing particularly on technical aspects and parallel scalability.

4.1.1 Lattice-Boltzmann on adaptive grids

On top of the steps described above, we implemented communication hiding in our ported version of ESPResSo's LBM solver for dynamically adaptive grids [56,58,72]. We overlap communication in the main loop after streaming with collision in local cells of a partition. The communication has to be finished before redundantly colliding in ghost cells. Another overlapping has been realized during grid adaptation. After repartitioning, we overlap transferring the numerical payload to their new owners with rebuilding `p4est` meta structures (`p4est_ghost`, `p4est_mesh`, `p4est_virtual`, and `p4est_virtual_ghost`). We use global instead of level-wise domain partitioning which yields good scalability[9] for our showcases but is meant to be replaced by level-wise partitioning in future work.

4.1.2 Electrokinetics and electrostatics on adaptive grids

The algorithms for charged systems have been implemented in [74]. To avoid interpolation errors for coupling subsystems and to reduce the number of different grid to

[8] Mass-based quantities are evenly distributed among the virtual cells and added up. Numeric quantities are mirrored on each virtual sub-cell and averaged.

[9] Presumably due to the fact that we avoid global synchronization within coarse grid time steps.

be considered for joint partitioning, we decided to use the same `p4est` instance as for the adaptive LBM. To cope with hanging nodes we embed virtual cells at refinement boundaries in electrokinetics. For electrostatics, we directly adapt the stencil based on the discretization level of the neighbors.

4.1.3 Short-range molecular dynamics with `p4est` linked-cell systems

In contrast to the grid-based algorithms described above, the computational load does not directly scale with the number of grid cells for short-range MD but with the number of force pairs. However, porting short-range MD to a `p4est`-based implementation using regular grids [75,76] allows performing MD-simulations with dynamic load-balancing. The main challenge consists of constructing octree grids with mesh widths as close as possible to the cut-off radius r_c. To this end, our implementation tries to accordingly fit the macro-structure, that is number and size of the individual trees in the forest of octrees. If an exact fit cannot be achieved, we choose r_c as a lower bound for the actual mesh width. This leads to more particle pairs being considered in neighboring cells, but ensures that we do not have to search for relevant particles within r_c in more cells than the direct neighbors of the current cell. Moreover, as described below, integrating `p4est` facilitates coupling short-range MD with the grid-based algorithms.

4.2 Coupled systems

To couple the individual physical components described in Section 2, ESPResSo provides a bi-directional force-coupling [77,78]. As we use at least two distinct `p4est` instances, that is a regularly discretized `p4est` for the Linked-Cell method in short-range MD and a potentially arbitrarily refined, yet 2:1 balanced `p4est` for the grid-based algorithms, we have to provide a mapping between both grids. Thus, we have to find a cell covering a given position. We calculate the position's (virtual) Morton index based on the maximum refinement level and the maximum domain size. Then, we perform a binary search on the local cells.

For actually mapping data, we perform tri-linear interpolation. Contrary to regular grids, neither the number of interpolation points nor the distance between cells used for interpolation is fixed. In case of 2:1 balanced grids, there are at most twenty cells involved: (1) the cell containing the position, (2) one corner neighbor, (3) up to twelve face neighbors, and (4) up to six edge neighbors. Additionally, the refinement pattern of the adaptive grid is not known beforehand. Thus, it is not sufficient to search one cell and obtain the remaining cells from index calculations, but we have to perform eight distinct binary searches [75].

To avoid communication and a volume-to-volume mapping-problem, we ensure that all information required for coupling is guaranteed to be found on the current processor. We achieve this by calculating a "finest common tree" (FCT) from all used `p4est` grid instances. Subsequently, we partition this FCT based on cell-weights that are derived from estimated cost weights depending on the involved model components and the depth of local further refinement of the respective component tree compared to the FCT. Then, we map the position of the partition boundaries back to each individual components' trees [79].

5 Results

5.1 Performance and scalability

We have published several studies on the performance and scalability of simulations in `p4est` in previous papers, which we summarize here pointing out the main lessons learned.

5.1.1 Regularly refined grids

In [56], we compare the `p4est` based implementation of the Lattice-Boltzmann method in ESPResSo with the original regular grid implementation for a Poiseuille flow scenario. All calculations were performed on a shared memory machine with 72 physical cores (Intel Xeon CPU E7-8880 @2.30GHz) and 512 GB RAM. We compared runtimes of Lattice-Boltzmann steps on regularly refined grids while still executing all functions required for dynamic grid adaptivity in the `p4est` variant: (1) evaluation of refinement and coarsening flags and re-generation of a new grid after each time step, (2) neighbor access via lookup tables in `p4est_mesh`, (3) more complex Morton curve based partitioning and corresponding ghost layer communication.

Simulations performed on 2–64 cores with one MPI rank per core and grids of refinement levels four to eight showed that the `p4est` variant was a factor of two to four slower than the original regular grid implementation. In additional test, we measured the pure grid traversal time in `p4est` with `p4est_iterate` and `p4est_mesh` and observed that we pay about a factor of two in terms of increased runtime for direct neighbor access. These results show a significant cost of dynamic grid adaptivity. However, they have to be interpreted carefully since usually dynamic grid changes are not executed after every time step and the implementation with `p4est` was an un-optimized version, e.g., still lacking communication hiding that was implemented later. Still, adaptivity always comes at a certain prize that has to be outweighed by a substantially reduced number of grid cells. Below, we show application examples where this is the case and total runtimes are dramatically lower than for corresponding regular grids.

5.1.2 Scalability of adaptive grid Lattice-Boltzmann

In [58], we present extensive strong and weak scaling studies on the Cray XC40 Tier 1 supercomputer HazelHen[10] at the High Performance Computing Center Stuttgart (HLRS). We simulate a three-dimensional driven cavity scenario, cf. Figure 3, and introduce artificial dynamic adaptivity by prescribing higher refinement in a moving rectangular subdomain. This allows us to control the number of grid cells, which is essential for weak scaling. Simulations were performed on up to 32 768 cores with one MPI rank per core.

The results, visualized in Figure 3, experimentally prove the efficiency of the Morton curve partitioning (with uniform weights for each cell) for the level-wise iteration of the space-time adaptive Lattice-Boltzmann steps. We show the total runtime of 10×16 integration steps and 10 grid adaptations with communication hiding in a weak scaling setting. We begin scaling from serial execution (red), single node execution (blue), and double node execution (black) and occupy each node with 24 processes. This yields a parallel efficiency above 50%. Communication can be almost completely hidden behind calculation as described in previous sections, which substantially speeds up the simulation.

[10] www.hlrs.de/systems/cray-xc40-hazel-hen/

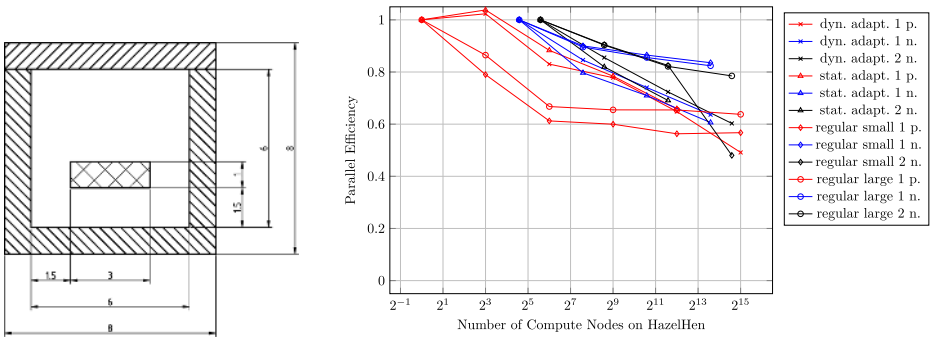

Fig. 3. A planar cut through the geometric setup of our driven cavity simulations is shown on the left. The parallel efficiency for pure Lattice-Boltzmann in a weak scaling setting measured on HazelHen is shown on the right. We use three distinct scaling bases (serial execution (red, "1 p."), single node (blue, "1 n."), and double node (black, "2 n.")) and compare four different cases: Two cases for regular grids and two adaptive cases. In the latter, the hatched box ($3 \times 3 \times 1.5$) marks an area of maximum refinement. The area is fixed in case of static adaptivity. In the dynamically adaptive case the area moves such that whenever the grid is adapted the area of maximum refinement has moved by one layer on the finest grid level. The runtimes of the respective base cases are shown in Table 1.

Table 1. Number of cells and total run times of the simulation in seconds for each distinct base case (serial execution, single compute-node, and two compute-nodes on HazelHen) of the driven cavity scenario.

Size	$n_{\text{cells, reg, s}}$	$t_{\text{reg, s}}$	$n_{\text{cells, reg, l}}$	$t_{\text{reg, l}}$	$n_{\text{cells, adapt}}$	$t_{\text{adapt, s}}$	$t_{\text{adapt, d}}$
1	65 536	35.66	52.4×10^5	309.98	1.48×10^5	161.62	161.79
24	5.24×10^5	17.32	4.19×10^6	137.35	1.07×10^6	58.96	62.83
48	65 536	1.12	5.24×10^5	8.86	8.12×10^6	223.82	234.63

Notes. We perform 160 time steps on the finest level. We show a small and a large regular setting ("reg, s" and "reg, l") as well as static and dynamically adaptive versions ("adapt, s" and "adapt, d"). In case of dynamical adaptivity, we adapt the grid and re-partition the cells after 16 time steps. The respective upscaling behavior is shown in Figure 3.

5.2 Coupled scenarios

We present new results for two scenarios to show the efficiency and validity of adaptive grid simulations in ESPResSo: a particulate flow through a simplified nanopore (coupling of short-range molecular dynamics and Lattice-Boltzmann) and an ionic flow in a more realistic pore geometry (coupling electrokinetics, electrostatics, and Lattice-Boltzmann).

5.2.1 Particulate flow through a nanopore

Following up on our scenario in [79], where we simulated particles in a simple channel flow, we simulate a *three-dimensional* pore scenario as illustrated in Figure 4. Initially, particles are randomly distributed in the left half of the domain, from where they are accelerated by a Lattice-Boltzmann flow from left to right. Our simulations are executed on the supercomputer HazelHen at the HLRS in Stuttgart with one MPI rank per core.

We perform a weak scaling study starting from a single compute-node (24 cores) to 64 nodes. On a single node, we simulate 24 000 particles. In each scaling step, we

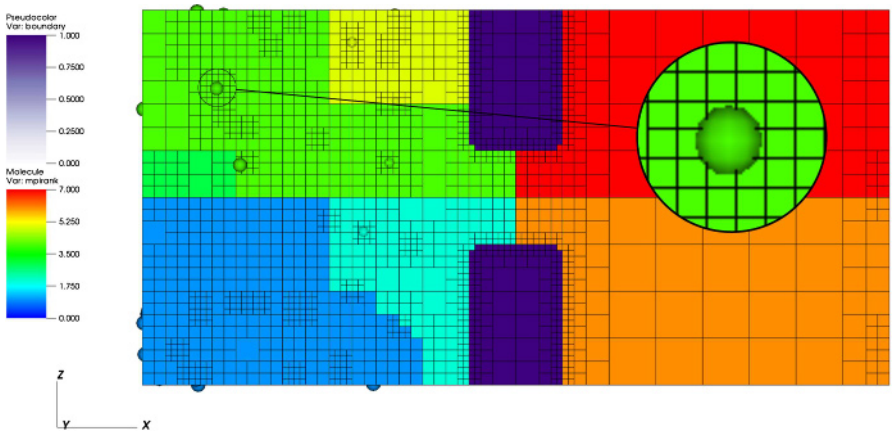

Fig. 4. Two-dimensional cut through the three-dimensional simulation setup of the particulate flow through a pore scenario. We place a simplified pore (purple) between two reservoirs and refine the grid around the pore as well as around the particles. Colors indicate the partitioning of computational domains among different ranks in the computational domain according to the Morton space-filling curve. Note that computational domains for particles and fluid are congruent to enable local coupling.

increase the minimum and maximum refinement level for the Lattice-Boltzmann flow solver by one. Accordingly, we increase the number of compute nodes and particles by a factor of eight, i.e., we simulate 24 000 to 1.5 million particles. The simulation domain is discretized with two connected trees, and we refine the grid around the boundary as well as around particles. All adaptive grids have a level difference between minimum and maximum of three. In *cases 3–6*, we scale from a grid with minimum and maximum levels three and six to minimum level five and maximum level eight (27×10^5 to 13.8 million cells), for *cases 4–7*, we go from levels four to seven to levels six to nine (1.8 million to 102.3 million cells). For comparison, we also measure the performance of stock ESPResSo and regular discretizations of our implementation. It is important to note that the original cpu implementation of the LBM in ESPResSo can only handle domains of the exact same size. Thus, we have to use sixteen cores as basis. Additionally, we use a slightly optimized LBM implementation which combines streaming and bounce back in a single grid traversal. This halves the costs of searching for neighbors in the LBM.

Results, averaged over three runs, are shown in Figure 5. We plot the total runtime as well as average runtimes for the main algorithmic substeps, that is LBM and MD time steps as well as altering the grid. Additionally, we visualize the Fluid-Lattice Updates per Second per Core (FLUPSC) and the imbalance in the LBM as maximum runtime over average runtime. For *cases 3–6*, we obtain a parallel efficiency of 50 % while we reach a parallel efficiency of over 90 % in the larger *cases 4–7*. ESPResSo's original LBM implementation reaches 1.7 MFLUPSC while our implementation reaches on average 0.2 MFLUPSC with merging of streaming and bounce back[11].

Moreover, we observe that load-balancing such a large simulation is challenging. We visualize the imbalance in LBM, MD, and adaptivity in a dynamically adaptive simulation on 1536 ranks in Table 2. This simulation consists of around 100 million LBM cells and 1.5 million particles. The imbalance in the LBM is approximately 1.3

[11] For comparison: the highly optimized code waLBerla achieves up to 5 MFLUPSC on SuperMUC [80] using pure MPI parallelization.

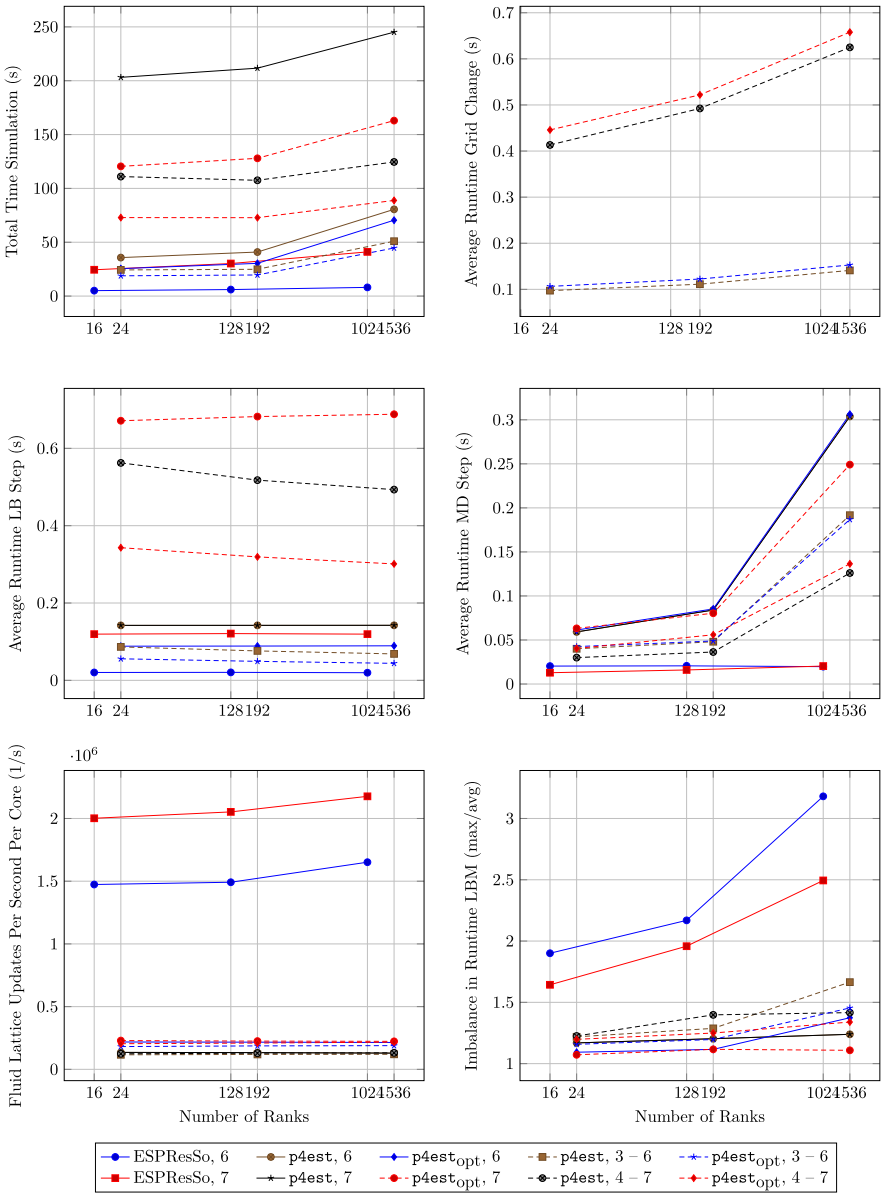

Fig. 5. Weak scaling runtime results for the pore scenario over 160 finest grid time steps. After 16 time steps, we adapt the grid of the LBM simulation according to the updated positions of the particles. We compare the previous ESPResSo (ESPResSo) version with our implementation (`p4est`) and a slightly optimized version (`p4est` $_{\text{opt}}$). The number indicates the refinement level of regular grids or the span of levels used in a dynamically adaptive discretization for the respective smallest problem of the scaling series. Values are averaged over ranks and time steps. Top left: total runtime; top right: runtime for grid changes; mid left: runtime of a single LBM step; mid right: runtime of a single MD step; bottom left: MFLUPSC; bottom right imbalance in LBM (max/avg).

Table 2. We visualize the imbalance for each time step for a run on 1536 cores using the optimized version of `p4est` on a dynamically adaptive grid using refinement levels between six and nine.

Time step	n_{cells}	imb. LBM	imb. MD	imb. grid change
t_0	1.067×10^8	1.291	7.259	1.002
t_1	1.013×10^8	1.295	1.135	1.002
t_2	1.012×10^8	1.302	1.218	1.003
t_3	1.012×10^8	1.292	1.069	1.002
t_4	1.012×10^8	1.301	1.136	1.003
t_5	1.012×10^8	1.284	1.111	1.002
t_6	1.012×10^8	1.301	1.05	1.002
t_7	1.012×10^8	1.284	1.338	1.002
t_8	1.012×10^8	1.3	1.254	1.002
t_9	1.013×10^8	1.304	1.144	1.002

Notes. We measure imbalance as maximum run time over average run time.

in all steps, while, for MD, it is also bounded by 1.4 except for the very first time step. Grid adaptivity is almost perfectly balanced. This behavior stems from using a grid library: When using uniform weights for all cells in our grids with strong refinement at boundaries, we obtain partitions which contain mostly boundary cells. This could be avoided by assigning boundary cells lower weights. Note that these weights should not be zero and have to be calibrated depending on the hardware architecture as, using a grid-library, operations such as neighbor-access and exchanging data between partitions is executed in all cells.

To improve overall performance, there are well-known optimizations like embedding small regular grids into octree cells that can be incorporated in our version in future work. Moreover, embedding patches facilitates using a streaming-optimized data-layout which is known to be beneficial in terms of cache efficiency [81].

By design, the average number of particles per core is 1000. The inhomogeneity of our setup leads to processes without any local particles on the one hand and processes with more than 3000 local particles in *cases 3–6* and up to 5000 local particles in *cases 4–7*. In *cases 3–6*, the number of particles in the ghost layer of the partitions varies between 0 and 2000. The upper bound increases in *cases 4–7* to 2500 ghost particles.

Note that adapting the grid includes deciding which cells to alter, actually changing the grid, establishing 2:1 balancing, mapping data between both grids, repartitioning based on the FCT, transferring the payload to its new owners, and rebuilding all `p4est` meta-structures such as `p4est_ghost` or `p4est_mesh`. Still, the complete grid adaptation never takes longer than 0.75 seconds.

5.2.2 Electrokinetic flow in a pore geometry

Based on the experimental setup of [82], our collaborators from project C.5 performed 2D simulations with the finite element method [83]. We ported the simulation setup of [83] to a three-dimensional model in ESPResSo.

In this setup, which is depicted in Figure 6, we use the full electrokinetic simulation, embedding ions and counterions into the fluid, applying an electric field by setting a potential at both hemispheres (orange and red boundaries), and charging the body of the capillary (royal blue boundary). This results in an electro-osmotic flow, as ions have accumulated around the pore to compensate the local charge. The

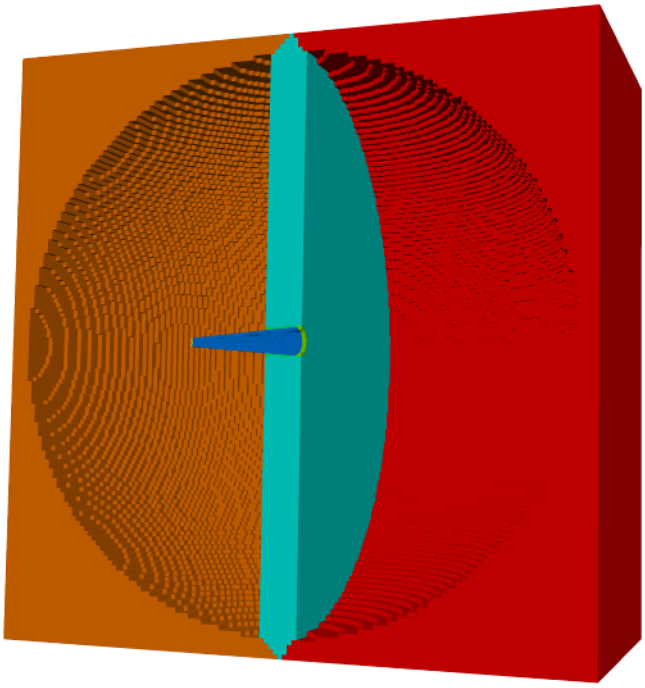

Fig. 6. Simulation setup of a charged pore. We charge the body of the capillary (royal blue) and apply an electric field by setting a potential at the hemispheres (orange and red) to obtain an electro-osmotic flow around the pore.

electric field that is applied from the hemispheres leads to a force density acting on the ions which is transferred to the fluid. In our preliminary simulation run, we have chosen a discretization from levels seven to ten and a pore diameter of 250 nm[12]. The resulting flow-field after 8000 time steps is shown in Figure 7.

6 Conclusion and outlook

We have presented ways to port a large scale multi-functional regular grid simulation code for a variety of different types of models (continuum mechanics and molecular dynamics) to tree-structured dynamically adaptive grids. We observed during the integration process, that it is highly beneficial to separate the actual grid functionality from the application code. In the work presented in this paper, we show how we extended the grid framework `p4est` to meet all requirements, in particular random access to direct neighbors, that we derive from the wish to keep application code changes as light-weight as possible. Besides the option to use adaptive grids, and, therewith, simulate substantially larger spatial domains and time spans, the application code ESPResSo in our example also profits from functionalities such as domain partitioning and inter-process communication in the parallel implementation, that previously had to be provided by the application code itself and can now be taken care of (in an optimized way) by the grid framework. Our results show the general suitabiltity and potential of our approach for various show-cases. We can show good

[12] We need a higher grid-resolution for embedding real or virtual particles into the given system as well as for modeling the actual pore geometry.

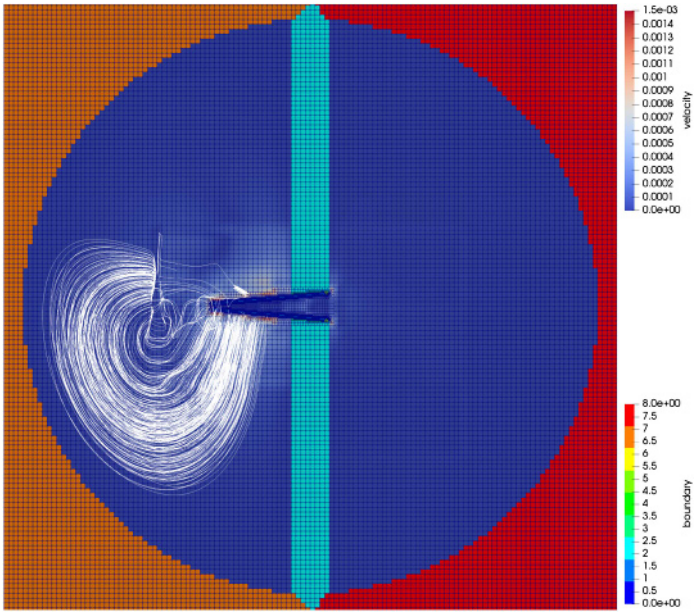

Fig. 7. Velocity field of the simulated electro-osmotic flow in the charged pore system.

parallel efficiency, low runtime overhead due to dynamical grid adaptivity, and a large potential to save computational cost for real-world simulation scenarios by grid adaptivity. Note that we are restricted in terms of code optimization by the fact that we keep code changes minimal and, in particular, do not alter or deteriorate parts of the existing extensive functionality of ESPResSo.

Future work, that has already begun is the implementation of a multi-GPU version of the Lattice-Boltzmann implementation in ESPResSo based on a patch-based approach. Preparatory steps have been made in the thesis work of Benjamin Kurz [84]. In addition, further optimizations such as a level-wise instead of global grid partitioning scheme [8] as well as more sophisticated grid cell weighting in the finest common tree partitioning are to be analyzed in terms of their effectiveness.

This work was financially supported by the German Research Foundation (DFG) via the grant SFB 716/D.8. Besides the authors, several other people have contributed to the work presented in this paper: Malte Brunn (IPVS, University of Stuttgart) has implemented a first version of the molecular dynamics simulation on `p4est` Linked-Cell systems and the coupling between molecular dynamics and Lattice-Boltzmann in his master thesis. Steffen Hirschmann (IPVS, University of Stuttgart, project D.9) has made substantial contributions in the supervision of this thesis, in optimizing the coupled molecular dynamics – Lattice-Boltzmann simulations and in setting up the particulate flow through a pore scenario. Georg Rempfer and Florian Weik (ICP, University of Stuttgart, project C.5) have provided helpful support in all questions concerning ESPResSo and the underlying physical models. Ingo Tischler has implemented the adaptive grid electrokinetics and electrostatics solvers in ESPResSo, jointly supervised by Michael Lahnert, Georg Rempfer, and Florian Weik. Carsten Burstedde (University of Bonn), the main developer of `p4est`, has contributed a lot of helpful advice on how to handle, extend, and optimize `p4est` and its use in ESPResSo.

References

1. T. Weinzierl, M. Mehl, SIAM J. Sci. Comput. **33**, 2732 (2011)
2. R.S. Sampath S.S. Adavani, H. Sundar, I. Lashuk, G. Biros, Dendro: parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees, in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (2008), pp. 1–12
3. C. Burstedde, L.C. Wilcox, O. Ghattas, SIAM J. Sci. Comput. **33**, 1103 (2011)
4. H. Klimach, K. Jain, S.P. Roller, End-to-end parallel simulations with APES, in *Advances in parallel computing* (IOS Press, 2014), Vol. 25, pp. 703–711
5. A. Lintermann, S. Schlimpert, J.H. Grimmen, C. Günther, M. Meinke, W. Schrüder, Comput. Methods Appl. Mech. Eng. **277**, 131 (2014)
6. T. Tu, D.R. O'Hallaron, 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)
7. F. Schornbaum, U. Rüde, SIAM J. Sci. Comput. **38**, C96 (2016)
8. F. Schornbaum, U. Rüde, SIAM J. Sci. Comput. **40**, C358 (2018)
9. S.G. Parker, Future Gener. Comput. Syst. **22**, 204 (2006)
10. M. Wahib, N. Maruyama, T. Aoki, Daino: A high-level framework for parallel and effcient AMR on GPUs, in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE, 2016), pp. 621–632
11. A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Lüffler, B. O'Shea, J. Parallel Distrib. Comput. **74**, 3217 (2014)
12. H.J. Limbach, A. Arnold, B.A. Mann, C. Holm, Comput. Phys. Commun. **174**, 704 (2006)
13. A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Roehm, P. Košovan, C. Holm, ESPResSo 3.1: Molecular dynamics software for coarse-grained models, in *Meshfree methods for partial differential equations VI*, Lecture Notes in Computational Science and Engineering, edited by M. Griebel, M.A. Schweitzer (Springer, Berlin, Heidelberg, 2012), Vol. 89, pp. 1–23
14. M. Bader, C. Böck, J. Schwaiger, C. Vigh, SIAM J. Sci. Comput. **32**, 212 (2010)
15. M. Bader, *Space-filling curves: an introduction with applications in scientific computing* (Springer, Heidelberg, New York, 2013)
16. G. Inci, A. Arnold, A. Kronenburg, R. Weeber, Aerosol Sci. Technol. **48**, 842 (2014)
17. J. Hüpfner, T. Richter, P. Košovan, C. Holm, M. Wilhelm, *Progr. Colloid. Polym. Sci.* **140**, 140 (2013)
18. B.J. Reynwar, G. Illya, V.A. Harmandaris, M.M. Müller, K. Kremer, M. Deserno, Nature **447**, 461 (2007)
19. M. Kuron, A. Arnold, Eur. Phys. J. E **38**, 20 (2015)
20. S. Kesselheim, M. Sega, C. Holm, Soft Matter **8**, 9480 (2012)
21. K. Breitsprecher, P. Košovan, C. Holm, J. Phys.: Condens. Matter **26**, 284108 (2014)
22. K. Breitsprecher, P. Košovan, C. Holm, J. Phys.: Condens. Matter **26**, 284114 (2014)
23. S. Succi, *The Lattice Boltzmann equation for fluid dynamics and beyond* (Clarendon Press, 2001)
24. D.A. Wolf-Gladrow, *Lattice-gas cellular automata and Lattice Boltzmann models – an introduction* (Springer, 2000)
25. S. Succi, *The Lattice Boltzmann equation: for complex states of flowing matter* (Oxford University Press, 2018)
26. P.L. Bhatnagar, E.P. Gross, M. Krook, Phys. Rev. **94**, 511 (1954)
27. D. d'Humières, S. Succi, D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, L.-S. Luo, Philos. Trans. R. Soc. A: Math. Phys. Eng. Sci. **360**, 437 (2002)
28. L.-S. Luo, W. Liao, X. Chen, Y. Peng, W. Zhang, Phys. Rev. E, **83**, 056710 (2011)
29. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
30. I. Ginzburg, F. Verhaeghe, D. d'Humieres. Commun. Comput. Phys. **3**, 519 (2008)
31. S. Hou, J. Sterling, S. Chen, G.D. Doolen, A Lattice Boltzmann subgrid model for high Reynolds number flow, in *Pattern formation and lattice gas automata* (American Mathematical Society, 1996), Vol. 6, pp. 151–166
32. R. Brownlee, A. Gorban, J. Levesley, Physica A **387**, 385 (2008)

33. X. Shan, X.-F. Yuan, H. Chen, J. Fuid Mech. **550**, 413 (2006)
34. R. Deiterding, S.L. Wood, J. Phys.: Conf. Ser. **753**, 082005 (2016)
35. M. Geier, A. Greiner, J.G. Korvink, Phys. Rev. E **73**, 066705 (2006)
36. P. Asinari, Phys. Rev. E **78**, 016701 (2008)
37. S. Seeger, H. Hoffmann, Continuum Mech. Thermodyn., **12**, 403 (2000)
38. M. Geier, M. Schönherr, A. Pasquali, M. Krafczyk, Comput. Math. Appl. **70**, 507 (2015)
39. E.K. Far, M. Geier, K. Kutscher, M. Krafczyk, Comput. Fluids **140**, 222 (2016)
40. K. Kutscher, M. Geier, M. Krafczyk, Comput. Fluids, in press
41. M. Geier, M. Schönherr, Computation **5**, 15 (2017)
42. M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Modeling and analyzing performance for highly optimized propagation steps of the Lattice Boltzmann method on sparse lattices, arXiv:1410.0412 (2014)
43. U.D. Schiller, Thermal uctuations and boundary conditions in the Lattice Boltzmann method, Ph.D. thesis, Johannes Gutenberg-Universität, Mainz, 2008
44. L. Li, R. Mei, J.F. Klausner, J. Comput. Phys. **237**, 366 (2013)
45. A. Pasquali, M. Geier, M. Krafczyk, Comput. Math. Appl., in press
46. A. Fakhari, T. Lee, Phys. Rev. E, **89**, 033310 (2014)
47. A. Fakhari, T. Lee, Comput. Fluids, **107**, 205 (2015)
48. O. Filippova, D. Hänel, J. Comput. Phys. **147**, 219 (1998)
49. D. Yu, R. Mei, W. Shyy, Int. J. Numer. Methods Fluids **39**, 99 (2002)
50. J. Tölke, S. Freudiger, M. Krafczyk, Comput. Fluids **35**, 820 (2006)
51. M. Geier, A. Greiner, J.G. Korvink, Eur. Phys. J. Special Topics **171**, 173 (2009)
52. H. Chen, Phys. Rev. E **58**, 3955 (1998)
53. H. Chen, O. Filippova, J. Hoch, K. Molvig, R. Shock, C. Teixeira, R. Zhang, Physica A **362**, 158 (2006)
54. M. Rohde, D. Kandhai, J.J. Derksen, H. Van den Akker, Int. J. Numer. Methods Fluids **51**, 439 (2006)
55. P. Neumann, Hybrid multiscale simulation approaches for micro- and nano ows, Ph.D. thesis, Technische Universität München, 2013
56. M. Lahnert, C. Burstedde, F. Weik, Towards Lattice-Boltzmann on dynamically adaptive grids – minimally-invasive grid exchange in ESPResSo. Englisch, in *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*, edited by M. Papadrakakis et al. (ECCOMAS, 2016)
57. M. Mehl, T. Neckel, P. Neumann, Int. J. Numer. Methods Fluids **65**, 67 (2010)
58. M. Lahnert, C. Burstedde, M. Mehl, Scalable Lattice-Boltzmann Simulation on Dynamically Adaptive Grids, submitted
59. F. Capuani, I. Pagonabarraga, D. Frenkel, J. Chem. Phys. **121**, 973 (2004)
60. G. Rempfer, Electrokinetic transport phenomena in soft-matter systems, Ph.D. thesis, University of Stuttgart, 2018
61. G. Rempfer, G.B. Davies, C. Holm, J. de Graaf, J. Chem. Phys. **145**, 044901 (2016)
62. P.P. Ewald, Ann. Phys. **369**, 253 (1921)
63. S.W. de Leeuw, J.W. Perram, E.R. Smith, Proc. R. Soc. London A: Math. Phys. Eng. Sci. **373**, 27 (1980)
64. R.W. Hockney, J.W. Eastwood, *Computer simulation using particles* (Taylor & Francis, Inc., Bristol, PA, USA, 1988)
65. M. Deserno, C. Holm, J. Chem. Phys. **109**, 7678 (1998)
66. M. Deserno, C. Holm, J. Chem. Phys. **109**, 7694 (1998)
67. A. Arnold, K. Breitsprecher, F. Fahrenberger, S. Kesselheim, O. Lenz, C. Holm, Entropy **15**, 4569 (2013)
68. J.E. Jones, Proc. Roy. Soc. London A: Math. Phys. Eng. Sci., **106**, 463 (1924)
69. J.E. Lennard-Jones, Proc. Phys. Soc., **43**, 461 (1931)
70. M. Buchholz, Framework zur Parallelisierung vonMolekulardynamiksimulationen in verfahrenstechnischen Anwendungen, Dissertation, München, Institut für Informatik, Technische Universität München, 2010
71. T. Isaac, C. Burstedde, L.C. Wilcox, O. Ghattas, SIAM J. Sci. Comput. **37**, C497 (2015)

72. M. Lahnert, T. Aoki, C. Burstedde, M. Mehl, Minimally-invasive integration of p4est in ESPResSo for adaptive Lattice-Boltzmann, in *The 30th Computational Fluid Dynamics Symposium* (Japan Society of Fluid Mechanics, 2016)
73. C. Burstedde, O. Ghattas, L.C. Wilcox, Towards adaptive mesh PDE simulations on petascale computers, in *Proceedings of Teragrid* (2008), Vol. 8
74. I. Tischler, Implementing adaptive electrokinetics in ESPResSo, MA thesis, University of Stuttgart, 2018
75. M. Brunn, Coupling of particle simulation and Lattice Boltzmann background flow on adaptive grids, MA thesis, Universität Stuttgart, 2017
76. S. Hirschmann, M. Brunn, M. Lahnert, C.W. Glass, M. Mehl, D. Pflüger, Load Balancing with p4est for Short-Range Molecular Dynamics with ESPResSo, in *Advances in parallel computing*, edited by S. Bassini et al. (IOS Press, 2017), Vol. 32, pp. 455–464
77. P. Ahlrichs, B. Dünweg, J. Chem. Phys. **111**, 8225 (1999)
78. B. Dünweg, A.J.C. Ladd, Adv. Polym. Sci. **221**, 89 (2009)
79. S. Hirschmann et al. Load-balancing and spatial adaptivity for coarse-grained molecular dynamics applications, in *High Performance Computing in Science and Engineering '18* (Springer, 2018), forthcoming
80. F. Schornbaum, Block-structured adaptive mesh refinement for simulations on extreme-scale supercomputers, Doctoral thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2018, p. 152
81. G. Wellein, T. Zeiser, G. Hager, S. Donath, Comput. Fluids, **35**, 910 (2006)
82. R.M.M. Smeets, U.F. Keyser, D. Krapf, M.Y. Wu, N.H. Dekker, C. Dekker, Nano Lett. **6**, 89 (2006)
83. G. Rempfer, S. Ehrhardt, C. Holm, J. de Graaf, Macromol. Theor. Simul. **26**, 160051 (2016)
84. B. Kurz, Lattice-Boltzmann simulationen auf mehreren GPUs, Bachelor's thesis, University of Stuttgart, 2018