Draft, July 2017

Load balancing with `p4est` for Short-Range Molecular Dynamics with ESPResSo

Steffen HIRSCHMANN [a,1], Malte BRUNN [a], Michael LAHNERT [a], Colin W. GLASS [b], Miriam MEHL [a] and Dirk PFLÜGER [a]

[a] *Institute for Parallel and Distributed Systems, University of Stuttgart, Germany*
[b] *High Performance Computing Center Stuttgart, University of Stuttgart, Germany*

Abstract. For short-range molecular dynamics (MD) simulations with heterogeneous particle distributions that dynamically change over time, highly flexible and dynamical load balancing methods are mandatory to achieve good parallel scalability. Designing and implementing load balancing algorithms is complex, especially for existing applications which were not designed to support arbitrary domain decompositions. In this paper, we present our approach to incorporate general domain decompositions and dynamic re-balancing into the existing MD software package ESPResSo. We describe the relevant interfaces and abstractions which enable us to reuse the physics algorithms in ESPResSo, without major re-implementations. As proof-of-concept, we show the implementation of a domain decomposition based on space-filling curves and a dynamic re-balancing mechanism using an enhanced version of the `p4est` library. The results indicate that our load balancing mechanism is capable of reducing the imbalance amongst processes and the total runtime of simulations in simple and complex scenarios. At the same time, the implementation of models and solvers in ESPResSo remains largely unchanged.

Keywords. Load balancing, distributed memory parallelization, domain decomposition, space-filling curves, molecular dynamics

1. Introduction

Short-range molecular dynamics (MD) [1,2] is an important simulation method in computational sciences. For example, in [3] it is used in conjunction with dynamic binding of particles to study the agglomeration of soot particles. To reduce the complexity of cal-

[1]Corresponding Author: Institute for Parallel and Distributed Systems, University of Stuttgart, Universitätsstraße 38, Stuttgart, Germany; E-mail: steffen.hirschmann@ipvs.uni-stuttgart.de.

culating the interactions for N particles to $\mathcal{O}(N)$, the so-called Linked-Cell method [4,5] is employed. The simulation of large amounts of particles still requires parallelization based on a domain decomposition approach, which is known to be scalable beyond other approaches for distributed memory parallelization, like atom or force decomposition [6]. However, we need sub-domains with more complex shapes than cuboids and suitable load balancing algorithms to address inhomogeneities of the particle distribution in space and time. In [7], we theoretically assess how to cope with inhomogeneities and identify relevant quantities for load balancing decisions. To verify these observations in real world applications, we present an implementation of one particular sophisticated domain decomposition and load balancing algorithm in ESPResSo. ESPResSo [8,9] currently uses rectangular domain decomposition and does not employ any kind of (dynamic) load balancing. In order to reuse all implemented physics of the code base, we aim to incorporate load balancing in a minimally invasive way. In this work, we explain the necessary interfaces for the integration of non-uniform domain decomposition and dynamical load balancing in existing short-range molecular dynamics simulation codes. Building upon these general concepts, we present a domain decomposition and load balancing method for ESPResSo based on space-filling curves (SFC). For the implementation, we utilize the existing `p4est` [10] library, which uses the Z-curve or Morton order [11] as an underlying principle for grid cell ordering in quad- or octree grids and as a domain partitioning tool. `p4est` provides a lot of efficiently implemented features required for domain partitioning such as the determination of cell neighborhoods and ghost layers of grid partitions.

Related Work. SFC-based decompositions are widely used for (adaptive) octree grids. A general introduction can, e.g., be found in [12]. Approaches to SFC-based decompositions for MD can, e.g., be found in [13]. The underlying idea is to distribute the computational load by splitting the linearization of the (linked) cells, given by the SFC, into contiguous sections with equal load. We describe the algorithm we use for partitioning in [7]. We base our implementation for SFC-partitioning on the `p4est` library [10] which provides a collection of algorithms and data structures for adaptive mesh refinement and mesh partitioning using octree grids (including grids composed of multiple trees). It is well-known to be scalable to up to hundreds of thousands of processes [14,15].

Other partitioning methods used in the literature are: Graph partitioning, see e.g. [16] for a model which incorporates dynamic re-balancing, and recursive bisection, see e.g. [17]. The latter is used in ls1 mardyn [18] and NAMD [19,20]. GROMACS [21] uses a similar, non-recursive approach. In contrast to these global algorithms (SFC-based partitioning and graph partitioning), local algorithms adapt an existing partitioning by only local exchanges of information and cells. They are, thus, considered to be inherently suitable for re-partitioning during runtime. Two prominent examples of local algorithms are diffusive strategies [22,23] and grid-based methods [24,25] which are used in IMD [26]. In our SFC-based partitioning, migration after re-partitioning is likely to happen only between a process and its predecessor and successor along the SFC. It therefore resembles a local algorithm in terms of communication requirements.

The authors in [27,28] conclude that SFC-based methods are efficient in terms of runtime and memory consumption, while generally giving a partitioning of good quality. Buchholz [29] draws the same conclusion for MD simulations in the MD-software ls1 mardyn on homogeneous and inhomogeneous scenarios. In his experiments, the SFC-based method and a recursive bisection approach usually yield the best partitioning. For example, the simulation of droplets composed of Lennard-Jones particles using an SFC-based decomposition is up to four times faster compared to a regular decomposition. This motivates that we test our concepts for a general domain decomposition with the SFC-based method. As mentioned before, we do this in ESPResSo which currently only supports a regular domain decomposition into equally sized rectangular sub-domains [9]. We use the `p4est` library to avoid the re-implementation of grid structures; the technical realization of the partitioning and neighborhood detection has already been integrated into ESPResSo in [30,31] for Lattice-Boltzmann simulations on adaptive grids.

Linked-Cell and Parallelization Basics & Interfaces. In molecular dynamics simulations with short-range interactions, the so-called Linked-Cell method [4,5] is typically used to identify particles within the cut-off radius of a given particle in constant time: All particles are binned into a grid with mesh width $h \geq r_{cut}$, where r_{cut} is the largest cut-off radius in the simulation, which is the maximum interaction range between particles. Thus, all interaction partners of a particle can be found in a neighborhood of 27 grid cells in three-dimensional simulations. The algorithmic building blocks defining the interfaces for the domain partitioning of a Linked-Cell grid implementation are: (1) Traversal of all local cells, (2) mapping of a particle position to the corresponding cell, (3) traversal of neighbor cells of a given cell, (4) traversal of all particles in a cell. To reduce the number of communication events, a domain decomposition requires introducing a so-called *ghost layer* around each sub-domain. It contains ghost cells mirroring boundary cells of neighboring processes such that all processes can calculate local interactions independently. Exchanging this information between different processes is called *ghost communication*. During the simulation, particles can leave their sub-domain and, thus, have to be migrated to the respective neighboring process. This step is called *particle exchange*. During particle exchange, all available information of migrated particles is exchanged, whereas ghost communication only exchanges the information required to compute particle interactions, i.e. positions and type. In summary, every process of a parallel deployment needs to (5) know which are its (geometrically) neighboring processes, (6) be able to map a particle position to the correct partitioning and corresponding process, (7) determine the ghost communication structure.

The structure of this work is as follows: In Section 2, we describe interfaces for parallelization based on a non-regular domain decomposition. In Section 3, we explain how we use `p4est` and how we achieve dynamic re-balancing in ESPResSo. Section 4 reports on results of the new domain decomposition and re-balancing. Finally, we summarize our work in Section 5 and conclude with a note on future research topics.

2. Non-Regular Domain Decomposition

We choose to use the existing interfaces in ESPResSo as much as possible to implement non-regular domain decomposition in a minimally invasive way. In the following, we outline the ESPResSo interfaces and our adaptions. The newly developed interfaces are applicable in ESPResSo, but were developed as a general template for domain decomposition and load balancing interfaces for existing MD simulation software.

Grid and Domain Decomposition. The current implementation of the Linked-Cell grid (“domain decomposition” in ESPResSo) is described in [9]. It is a general interface for a grid whose cells hold particles. ESPResSo stores the cells in a one-dimensional

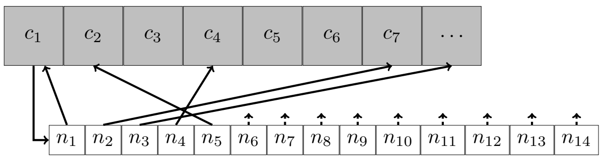

Figure 1. In gray: One-dimensional storage of cells. For cell c_1, the implementation of the neighborhood as a list of pointers or indices is shown (the first five neighbors only). Note that 14 is the number of half-shell neighbors in 3D.

array. Furthermore, each cell has its own list of neighboring cells (see Figure 1). This comes in handy, as it can be used to represent non-box-shaped subdomains, where, in addition, the numbers of inner and ghost cells are not known a-priori. Reusing internal data structures of ESPResSo or other existing MD solvers is essential for our minimally invasive integration, as all physics algorithms implemented in the code at hand depend on these core data structures. In conclusion, to comply with the ESPResSo interfaces, an underlying grid implementation needs to provide the number of inner and ghost cells as well as the cell neighborhood information. The mappings of particle positions to cell and particle positions to process depend on the organizational structure of the used grid and domain partitioning and, therefore, specific functions need to be implemented for every grid- and partitioning-type.

Communication. The dimension-wise, synchronized communication scheme [6] used in ESPResSo exploits a regular communication structure (two neighbors for each process in every coordinate direction). Since we assume no specific structure in the shape of subdomains, we also can not assume such a structure anymore. Therefore, we implement asynchronous ghost communication and particle exchange, which work for subdomains of arbitrary shape.

Particle Exchange. To migrate particles, we traverse all local particles and determine if a local particle left its subdomain. If so, the particle position corresponds to a ghost cell, of which we know the owner's rank. For some features, it is not guaranteed that all out-of-subdomain particles reside in ghost cells. For this, we use the global 'position to process' mapping. The communication scheme is as follows: First, an asynchronous receive operation for each neighbor is posted to receive the number of particles to be expected. Then, out-of-subdomain particles are sorted into the corresponding send buffers. Afterwards, the following data are sent to all neighbors: (1) The particle count, (2) the particles themselves (if any), and (3) dynamic data associated with these particles (e.g., bond lists). After sending these data, a process waits for the particle counts from each process to arrive and then posts asynchronous receive operations for particles and dynamic data.

Ghost communication. The ghost communication routine needs to know which ghost cells to receive from which neighbor as well as which boundary cells to send to which neighbor. To this end, ESPResSo provides a structure called `GhostCommunication` which consists of an operation (`send` or `receive`), a rank, and a list of cells to be sent or received. The specification of one ghost communication step consists of a list of `GhostCommunications`. This interface can be used for communications on arbitrarily structured domains, see Figure 2. However, the routine which performs the actual communication assumes that the domain is decomposed regularly into box-shaped subdomains and that the `GhostCommunications` are ordered such that a synchronous

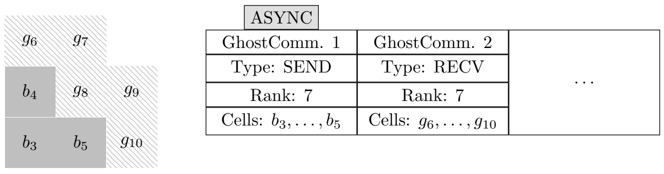

Figure 2. A segment of a non-regularly shaped subdomain with 6 ghost cells and 3 boundary cells. Exemplary, a list of two GhostCommunications is shown, the first a send operation on three boundary cells and the second a receive operation of six ghost cells. The list is tagged as being asynchronous.

communication does not cause deadlocks. We provide a ghost communication function which works asynchronously and accepts arbitrary communication specifications, such that it can be used in combination with non-box-shaped subdomains and an arbitrary amount of neighboring processes. Therefore, we extend the interface by introducing an `asynchronous` flag. If a list of `GhostCommunications` is tagged as asynchronous (as depicted in Figure 2) ESPResSo dispatches to our newly created asynchronous communication function. It works analogously to the communication scheme sketched above.

Determining the communication volume. Ghost communication includes communication across periodic domain boundaries. In ESPResSo, positions of particles involved in this case are shifted to the corresponding periodic image of the simulation box. As a consequence, however, if two processes share periodic domain boundaries in different directions, a single cell may need to be replicated multiple times at a unique neighbor process (multiple images). Note that this means that on the one hand the receiver has multiple distinct ghost cells, which are identical on the sending process and that, on the other hand, the sending process has to determine how often and with which shifts a neighbor process expects to receive the boundary cells. In order not to change how ESPResSo handles the calculation of particle distances, we have to account for these possibly multiple copies of the same cell and provide shift values. ESPResSo associates a shift value with each `GhostCommunication` which the sender adds to all outgoing particles. The shift value itself is, therefore, either 0, the box length or the negative box length. To generalize this concept, we iterate over all boundary cells and each of these cells' respective neighborhood. If the neighbor is a ghost cell, we determine whether and in which direction the two cells are separated by a periodic boundary and tag the boundary cell with this direction (possibly none) and the rank of the neighboring process. These direction-rank pair tags for each boundary cell are sufficient to fill the communication send lists.

3. Implementation of an SFC-based Domain Decomposition

In this section, we describe the implementation of an SFC-based domain decomposition in ESPResSo using `p4est` as a first proof-of-concept for our interfaces for non-regular subdomains and dynamic re-balancing. The use of `p4est` may seem like an overkill, as it provides full functionality for adaptively refined grid whereas our Linked-Cell grids are always regular, however, `p4est` not only provides convenient functionalities, but also the basis for coupling MD and Lattice-Boltzmann simulations in future work. We use `p4est` as a tool to populate the existing data-structures in ESPResSo as described in Section 2. In addition, we implement a number of auxiliary algorithms within ESPResSo complementary to p4est's functionalities, e.g., the mapping of an arbitrary particle position within the simulation domain to a specific process without using communication.

In order to minimize the amount of unnecessary distance calculations in the Linked-Cell algorithm, the mesh width h of the grid has to be as close as possible to the cut-off radius r_{cut}. Given a direction i and the respective domain length l_i, the standard number of cells in this direction is $n_i := \lfloor l_i / r_{cut} \rfloor$. This ensures that $h_i = l_i / n_i \geq r_{cut}$. Usually, these n_i are not powers of two. This, however, is required as a spatial discretization by p4est with a single octree. Therefore, we make use of p4est's feature to combine several octrees to one grid (forest of octrees): We determine the highest common divisor d of n_0, n_1, and n_2 that is a power of two, i.e., $d = 2^e$ with a suitable exponent e. With this, the number of necessary trees in direction i is $t_i := n_i / d$. In our dynamic load balancing approach, the initial domain decomposition is purely based on counting cells per partition. While the simulation proceeds, we collect data on the actual computational load of grid cells and use those as a weighting function. This allows us to dynamically adapt the partitioning towards an optimal load balancing.

To populate ESPResSo's internal neighbor lists, we first construct the ghost-layer of each process using p4est_ghost [15]. After that, we encode the process-local neighbor relations in p4est_mesh [31]. This information is sufficient to populate the existing data-structures, including copying and shifting duplicated ghost cells at periodic boundaries.

To implement the auxiliary position to cell and rank mappings, we overlay the domain with a single virtual octree with a regular refinement pattern. To map a position to a process, we perform a binary search over the indices of the first quadrant on each process, i.e., the indices defining subdomain boundaries, in the virtual octree. Searching a specific quadrant follows the same basic idea. We map the position of the respective quadrant to an index in the virtual octree and perform binary search on these indices.

Dynamic Re-balancing. To perform dynamic re-balancing during runtime we implement a new command repart in ESPResSo which can be called at any time. It takes a string describing the metric (function that gives each cell a weight) used for re-partitioning. We implement several different metrics: Number of particles per cell, distance pairs and force pairs to be calculated for particle interactions, bonded interactions and measured runtime. Any linear combination of these is valid as an overall partitioning metric. After evaluating the weights per cell, the new partition boundaries are determined by the SFC algorithm described in [7]. With this information, we call p4est_partition_given which re-partitions in p4est. Afterwards, we need to recreate p4est_ghost and p4est_mesh and re-initialize the cell system of ESPResSo adapting its internal data structures, cell grid, and GhostCommunications to the new partitioning. With the information about the old and the new partitioning, we implement a third communication routine, an optimized cell-based migration. Since partitions are plain intervals on the SFC, we can use interval intersection operations to determine if and how many cells a process needs to send or receive. This decision can be taken locally at each process. Given the old interval I_p and the new one I'_p along the curve, the send volume from process p to q is $I_p \cap I'_q$ whereas the receive volume is $I'_p \cap I_q$. The cells to migrate are, again, asynchronously communicated according to the communication scheme sketched above.

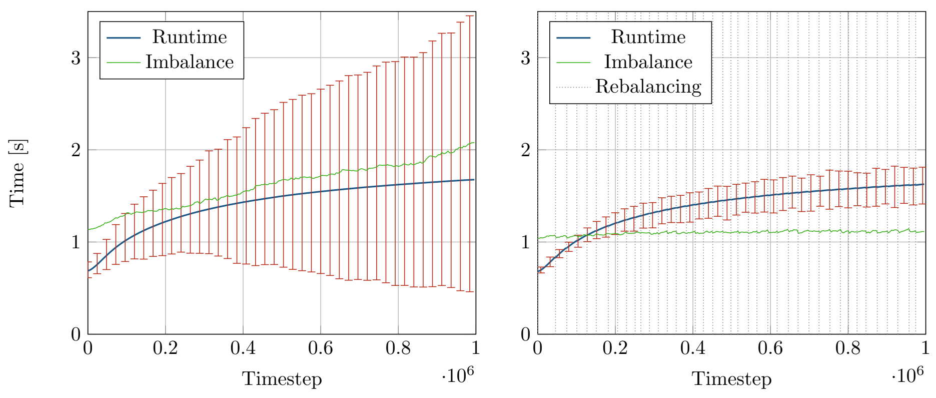

Figure 3. Runtime in seconds for 1000 successive timesteps each and imbalance for the force calculation. Left: default ESPResSo, right: our version. The error bars indicate the maximum and minimum runtime, respectively, while the blue line indicates the average. Note that for the overall performance the maximum runtime is the critical factor. The imbalance is shown in green. The vertical dashed lines in the right picture indicate time steps at which re-balancing is performed.

4. Results

In this section, we present results of the new domain decomposition and its dynamic re-balancing capabilities. First, we verify that the changes we introduced have no significant adverse affect on the runtime of an unbalanced simulation. Then we check the cost of the new domain decomposition on a homogeneous simulation: it scales slightly better than the original version of ESPResSo. Also, the runtime itself is lower if at least 8 nodes are used. This means that the benefit of asynchronous communication outweighs the run-time overhead of our new grid creation and the more complex position to cell mapping which is used for particle exchange during runtime. The force calculation itself remains unchanged, only the ordering of the cells and the subdomains are different. These results were obtained from a weak scaling experiment on the Tier-1 supercomputer “Hazel Hen” at the High Performance Computing Center Stuttgart (HLRS). It is a Cray XC40 installation consisting of 7712 computational nodes linked via Cray Aries interconnect. Each node is equipped with 24 Intel Haswell cores and 128 GB of RAM.

We also conduct two tests for heterogeneous and dynamical systems with re-balancing on “Hazel Hen”. The first scenario consists of 1700 initially homogeneously (but randomly) distributed Lennard-Jones particles with density (particle to volume ratio) 0.1. In this setup, particles start to form small droplets. The density of these droplets is higher and, thus, the load imbalance across processes of a uniform domain partitioning is slowly increasing during runtime. We simulate the fluid over the first 10^6 time steps and measure the execution times of force calculation, communication and time stepping plus synchronization accumulated over 1000 timesteps. Using the number of distance pairs per cell as metric, we re-partition if the imbalance (maximum divided by average) between the processes is higher than 1.1. Figure 3 shows the cost of the Linked-Cell algorithm for force calculation and its imbalance between the processes for the unbalanced, default version of ESPResSo and the balanced, `p4est`-based version. As we can see, even in this simple scenario, we can save calculation time and we effectively keep the imbalance down at a level close to 1. However, the distance pairs metric does not

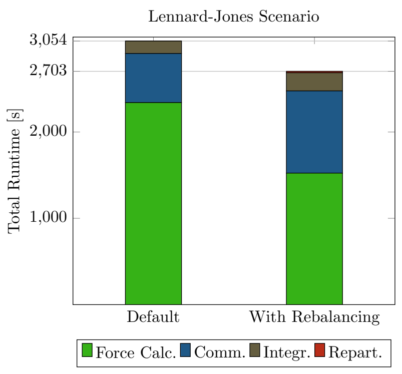

Figure 4. Runtime broken down into its components for the Lennard-Jones scenario. Note that “Integration” refers to all computations except for the force calculation and *synchronization*. Left: Default version of ESPResSo, right: Our version with dynamic re-balancing. The time required for re-balancing is only about 17 seconds in total.

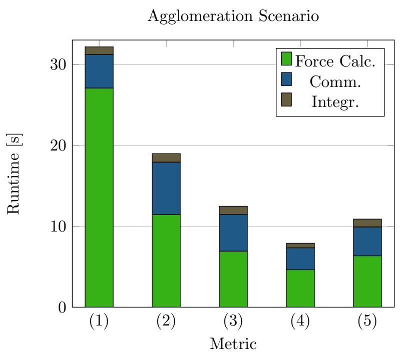

Figure 5. Runtime after re-balancing of the agglomeration scenario for 1000 timesteps. The different bars show different metrics used: (1) No re-partitioning, i.e. each cell uniformly weighted, (2) number of distance pairs, (3) number of force pairs, (4) number of particles and (5) number of bonded interactions.

take into account costs other than force calculations. We can see this by breaking down the total runtime of the simulation into the contributions of force calculation, communication and time stepping plus synchronization, see Figure 4. This drastically reduces the performance we can gain in such a simple scenario and shows that we should not disregard the communication cost. This is especially relevant for scenarios where the communication versus computation ratio is high. We can also see that the cost of re-balancing (approximately 17 seconds for all re-balancing steps) is negligible in this scenario.

To test re-balancing in a more complex scenario, we use ESPResSo’s dynamic binding and bonded, harmonic interaction features [8] in conjunction with a Lennard-Jones fluid. For dynamic binding of particles, we use the “All Bonds Model” from [32]. The simulation has 3.2 million particles and the setup is adapted from [3]. From this simulation, we take a snapshot and simulate it for 1000 timesteps, take measurements as described above, re-partition it with different metrics and simulate it, again, for 1000 timesteps. The results are shown in Figure 5. As we can see, the optimal choice of a metric is very important in this example. The number of LJ particle pairs performs worst amongst the tested metrics. This is due to the different algorithmic costs. First, the cell size has to be increased in order to transfer all possible bond partners of local particles in the ghost exchange. Furthermore, for all possible LJ force pairs, a collision detection is performed. These facts make the LJ force pairs metric perform better than the LJ distance pairs metric. Additionally, there are costs associated with the bonded interactions. These are highly relevant as partitioning based only on the number of bonded interactions performs better than based on LJ force pairs: The harmonic bonds are more costly to calculate than simple LJ interactions. However, the best partitioning amongst all tested is achieved by simply using the number of particles per cell as a metric. Note, that we assume that there exists a linear combination of the metrics used here which performs better than any single one. Also note, that unlike with the number of distance pairs as metric in this scenario and the Lennard-Jones scenario, the communication cost does not increase with the number of particles as metric.

5. Conclusion

We have presented a minimally invasive way to implement an arbitrary (non-regular) domain decomposition and dynamic re-balancing into an existing MD simulation software. We implemented the concepts in ESPResSo for an SFC-based domain decomposition using the `p4est` library as a proof-of-concept. We have shown relevant interfaces and abstractions. We have sketched the current implementation on ghost communication in ESPResSo and how we reuse ghost communication and data structures from ESPResSo. We have described the necessity and the implementation of asynchronous communication.

We have presented tests on scaling and re-balancing. The new implementation scales better than the version of ESPResSo which uses a domain decomposition into equally sized boxes on a Cartesian grid, while exact numbers depend strongly on the scenario. Finally, we have demonstrated that the new dynamic re-balancing has the potential for gains in runtime if employed correctly. We also have shown that (especially considering SFCs), the underlying algorithmic physics models need to be known and incorporated into the metric for re-balancing to achieve good re-balancing.

Future Work. Based on the results for different metrics we got in Section 4, the question of the optimal metric for a given scenario arises. We plan to address this in the future and try to make it possible to determine good load-balancing schemes for scenarios in general. The other open issue shown in this section was the cost of communication. Incorporating communication cost into the SFC-based method is not trivial, however, as the boundary is given implicitly and the costs for moving cells cannot be attributed to cells but depend on the overall partitioning.

Another future task we aim for is to provide load balancing and metrics in case of coupled simulations, in particular coupling ESPResSo with the Lattice-Boltzmann method (LBM) [31]. MD and LBM are both available in the default version of ESPResSo. Finally, our goal is to implement more of the load balancing methods mentioned in the related work, to be able to compare between different methods and devise good overall load balancing strategies.

Acknowledgements. The authors gratefully acknowledge funding provided by the German Research Foundation (DFG) as part of the Collaborative Research Center (SFB) 716 and the computing resources provided by the High Performance Computing Center Stuttgart (HLRS). The authors also like to thank Carsten Burstedde for fruitful discussions and valuable input on `p4est`.

References

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

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

[3] G. Inci et al. Langevin dynamics simulation of transport and aggregation of soot nano-particles in turbulent flows. *Flow, Turbulence and Combustion*, pages 1–21, January 2017.

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

[5] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids*. Oxford Science Publ. Clarendon Press, 1989.

[6] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.

[7] S. Hirschmann et al. Towards understanding optimal load-balancing of heterogeneous short-range molecular dynamics. In *Workshop on High Performance Computing and Big Data in Molecular Engineering 2016 (HBME 2016)*, Hyderabad, India, December 2016.

[8] A. Arnold et al. ESPResSo 3.1: Molecular dynamics software for coarse-grained models. In *Meshfree Methods for Partial Differential Equations VI*, volume 89 of *Lecture Notes in Computational Science and Engineering*, pages 1–23. Springer Berlin Heidelberg, September 2012.

[9] H. J. Limbach et al. ESPResSo – an extensible simulation package for research on soft matter systems. *Computer Physics Communications*, 174(9):704–727, May 2006.

[10] C. Burstedde et al. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

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

[12] P. M. Campbell et al. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, Williamstown, MA 01267, January 2003.

[13] C. Xiaolin and M. Zeyao. A new scalable parallel method for molecular dynamics based on cell-block data structure. In *Parallel and Distributed Processing and Applications*, volume 3358 of *Lecture Notes in Computer Science*, pages 757–764. Springer Berlin Heidelberg, 2005.

[14] J. Rudi et al. 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*, pages 5:1–5:12. ACM, 2015.

[15] T. Isaac et al. Recursive algorithms for distributed forests of octrees. *SIAM J. Sci. Comput.*, 37(5):C497–C531, January 2015.

[16] U. V. Catalyurek et al. A repartitioning hypergraph model for dynamic loadbalancing. *J. Parallel Distrib. Comput.*, 69(8):711–724, August 2009.

[17] F. Fleissner and P. Eberhard. Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. *International Journal for Numerical Methods in Engineering*, 74(4):531–553, 2008.

[18] C. Niethammer et al. ls1 mardyn: The massively parallel molecular dynamics code for large systems. *J. Chem. Theory Comput.*, 10(10):4455–4464, October 2014.

[19] L. Kalé et al. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151(1):283–312, 1999.

[20] A. Bhatelé et al. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 110–116, New York, NY, USA, 2009. ACM.

[21] B. Hess et al. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *J. Chem. Theory Comput.*, 4(3):435–447, February 2008.

[22] J. E. Boillat et al. A dynamic load-balancing algorithm for molecular dynamics simulation on multi-processor systems. *Journal of Computational Physics*, 96(1):1–14, 1991.

[23] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel Distrib. Syst.*, 4(9):979–993, September 1993.

[24] Y.-F. Deng et al. Molecular dynamics on distributed-memory MIMD computers with load balancing. *Applied Mathematics Letters*, 8(3):37–41, 1995.

[25] Y. Deng et al. An adaptive load balancing method for parallel molecular dynamics simulations. *Journal of Computational Physics*, 161(1):250–263, 2000.

[26] C. Begau and G. Sutmann. Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations. *Computer Physics Communications*, 190(0):51–61, 2015.

[27] S. Schambeger and J. M. Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *Proceedings of the 7th International Conference on Parallel Computing Technologies*, volume 2763 of *LNCS*, pages 165–179, 2003.

[28] W. F. Mitchell. A refinement-tree based partitioning method for dynamic load balancing with adaptively refined grids. *J. Parallel Distrib. Comput.*, 67(4):417–429, April 2007.

[29] M. Buchholz. *Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen*. Verlag Dr. Hut, 2010.

[30] M. Lahnert et al. Minimally-invasive integration of p4est in ESPResSo for adaptive Lattice-Boltzmann.

In *The 30th Computational Fluid Dynamics Symposium*. Japan Society of Fluid Mechanics, 2016.

[31] M. Lahnert et al. Towards Lattice-Boltzmann on Dynamically Adaptive Grids – Minimally-Invasive Grid Exchange in ESPResSo. In *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*. ECCOMAS, June 2016.

[32] G. Inci et al. Modeling nanoparticle agglomeration using local interactions. *Aerosol Science and Technology*, 48(8):842–852, July 2014.