Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Physics Communications 174 (2006) 704–727

Computer Physics Communications

www.elsevier.com/locate/cpc

ESPResSo—an extensible simulation package for research on soft matter systems

H.J. Limbach [a,*,1], A. Arnold [a,b], B.A. Mann [a], C. Holm [a,b]

[a] *Max-Planck-Institut für Polymerforschung, Ackermannweg 10, D-55128 Mainz, Germany*
[b] *Frankfurt Institute for Advanced Studies (FIAS), JW Goethe-Universität Frankfurt, Max-von-Laue-Strasse 1, D-60438 Frankfurt am Main, Germany*

Received 18 April 2005; received in revised form 9 September 2005; accepted 24 October 2005

Available online 20 March 2006

Abstract

We describe a new program package that is designed to perform numerical Molecular Dynamics (MD) and Monte Carlo (MC) simulations for a broad class of soft matter systems in a parallel computing environment. Our main concept in developing ESPResSo was to provide a user friendly and fast simulation tool which serves at the same time as a research platform capable of rapidly incorporating the latest algorithmic developments in the field of soft matter sciences. A particular strength of ESPResSo is its efficient treatment of long range interactions for various geometries using sophisticated algorithms like P^3M, MMM2D, MMM1D and ELC. It is already equipped with a broad variety of interaction potentials, thermostats, and ensemble integrators; it offers the usage of constraints, masses and rotational degrees of freedom; it allows to move between different ensembles on-the-fly. An efficient MPI parallelization allows the usage of multi-processor architectures. Strict usage of ANSI-C for the core functions and a Tcl-script driven user interface makes ESPResSo platform independent. This also ensures easily modifiable interfaces to communicate with other MD/MC Packages, real-time visualization and other graphic programs. We tried to maintain a clear program structure to keep ESPResSo extensible for future enhancements and additions. ESPResSo is implemented as an open source project with the goal to stimulate researchers to contribute to the package.

PACS: 05.10.-a; 87.15.Aa; 02.70.Ns; 07.05.Tp

Keywords: Computer simulation; Parallel Molecular Dynamics; Benchmarks

PROGRAM FEATURES

Feature	Exemplary Tcl-command
Integrators/Thermostats	
(N, V, E)-ensemble	`thermostat off; integrate 1000`
(N, V, T)-ensemble	`thermostat langevin 1 1; integrate 1000`
(N, p, T)-ensemble	`thermostat npt_isotropic 1 1 1; integrate 1000`
Nonbonded potentials	
Lennard-Jones	`inter 0 1 lennard-jones 1 1 1.12246 0.25 0`
Morse	`inter 0 1 morse 1 1 1.12246 0.25 0`
Buckingham	`inter 0 1 buckingham 1 1 1 1 1.5 1 0.25`

[*] Corresponding author.
E-mail addresses: hans-joerg.limbach@rdls.nestle.com (H.J. Limbach), arnolda@mpip-mainz.mpg.de (A. Arnold), mann@mpip-mainz.mpg.de (B.A. Mann), holm@mpip-mainz.mpg.de (C. Holm).
[1] Present address: Food Science Department, Nestlé Research Center, Vers-chez-les-Blanc, P.O. Box 44, CH-1000 Lausanne 26, Switzerland.

0010-4655/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2005.10.005

Soft sphere	`inter 0 1 soft-sphere 1.2 1 1.5 0`
LJ + cosine	`inter 0 1 ljcos 1 1 1.5 0`
Gay–Berne	`inter 0 1 gay-berne 1 0.5 0.5 1.5`
tabulated	`inter 0 1 tabulated mylj.dat`
Bonded potentials	
FENE	`inter 0 fene 10 1`
harmonic	`inter 0 harmonic 10 1`
angle	`inter 0 angle 10`
dihedral	`inter 0 dihedral 1 10 0`
tabulated	`inter 0 tabulated bond myfene.dat`
Long-ranged potentials	
3D Coulomb/P3M	`inter coulomb 1 p3m tunev2 accuracy 1e-4`
2D Coulomb/MMM2D	`inter coulomb 1 mmm2d 1e-4`
2D Coulomb/ELC	`inter coulomb elc 1e-4 10`
1D Coulomb/MMM1D	`inter coulomb 1 mmm1d tune 1e-4`
3D Coulomb/Maggs	`inter coulomb 1 maggs 0.1 32 1`
General analysis	
energy	`analyze energy`
pressure	`analyze pressure`
radial distribution	`analyze rdf 0 10 0.1`
structure factor	`analyze structurefactor 0 10`
Polymer analysis	
end-to-end distance	`analyze re 0 30 200`
radius of gyration	`analyze rg`
$g_{1,2,3}$	`analyze g123`
File I/O	
ESPResSo blockfiles	`blockfile $file write particles id pos v`
blockfile trajectories	`checkpoint_set ’’run_1’’`
PSF/PDB	`writepsf ’’config’’; writepdb ’’config’’`
Other	
VMD visualization	`prepare_vmd_connection ”test”; imd positions`

1. Introduction

Soft matter is a term for materials in states that are neither simple liquids nor hard solids of the type studied, for example, in solid state physics. Examples are polymers, colloids, liquid crystals, glasses, and dipolar fluids. Many such materials are familiar from everyday life—glues, paints, soaps, baby diapers—while others are important in industrial processes, such as polymer melts that are molded and extruded to form plastics [1], or polymer networks for the development of rubber [2]. Biological materials are mainly made out of soft matter as well—DNA, membranes, filaments and other proteins belong to this class. Furthermore, most of the food we digest is soft matter.

All these materials have in common that a wide range of length and time scales is important for their microscopic behavior as well as their macroscopic properties. Typical energies between different structures are similar to thermal energies. Hence, Brownian motion or thermal fluctuations play a prominent role. Another key feature of soft matter systems is their propensity to self-assemble. This often results in complex phase behaviors yielding a rich variety of accessible structures. Most of the biological systems are usually not even in equilibrium but evolve among switchable steady states. Many of the soft matter systems are highly complex in their composition and their interactions, and therefore they are also sometimes termed complex fluids. Much of today’s insight has been gained through computer simulations of these systems at either the full atomistic level or on a coarse grained, so-called mesoscopic, level.

In the past, our research has mainly focused on the study of charged polymers (polyelectrolytes) and charged colloids which serve as important substances for many technical applications. Charged systems also occur in biological environments [3] (since most biological matter is charged), and modeling explicit water molecules requires partial charges as well. The simulation of these systems is not straightforward and very time consuming [4–6], since the effort scales at best linearly with the number of charges, thus the production of single data points could take weeks or even months for complex biomolecular problems [7]. Consequently, we have improved or developed a number of algorithms which yield fast expressions for the energy and forces of fully or partially periodic systems [8–12]. We are also interested in simulating such systems in various geometries, e.g., films or porous media and with different topologies ranging from linear to branched to networks [13,14]. Another focus is to systematically coarse grain atomistic models such that one can access mesoscopic time and length scales with computer simulations [15–17]. This is necessary to predict quantitatively dynamic properties or surface interactions for complex materials which depend on local chemical interactions and local packing. For the coarse graining procedure it is required to switch back and forth between an atomistic and a coarse grained description. This implies complicated steering schemes for a simulation. Therefore, we need a high flexibility of the steering level and an easy way to interact with other programs. Therefore the desired program has to be extensible such one can easily implement and test new methods and algorithms. These are usually tasks that involve changes in the core part of the program.

Looking at other available simulation packages, e.g., BALL [18], GISMOS [19], GROMOS [20], GROMACS [21], LAMMPS [22], NAMD [23,24], polyMD [25], Amber [26], NWChem [27], DL_Poly [28], and OCTA [29], we did not find a single package which met all our needs. It should be easy to use, but scientifically sound; it should grant experts access to state-of-the-art techniques, but enable beginners to investigate scientific problems easily and at the same time not limit them to run the program as a “black box”; it should be general and flexible, but still fast; it should be easily extensible, but remain at the same time reliable and keep continuity with older versions. This led us to design a newly structured program for this field of science, which we called an **E**xtensible **S**imulation **P**ackage for **Res**earch on **So**ft Matter Systems, ESPResSo for short, see Ref. [30] for an earlier account. Note that in the meantime another project called Quantum-Espresso [31] has been established, which is a package for Car–Parinello MD simulations and should not be confused with our program, despite the similar names.

Our program is designed to study soft matter model systems via Molecular dynamics (MD) algorithms, with particular emphasis on extensibility for new, highly complex force/energy algorithms. ESPResSo is parallelizable and fast and thus able to compete in speed with any of the above mentioned programs (see Section 6). On the steering level it is far more flexible than any other simulation tool with a similar scope that we have seen. Still the clear program structure together with the documentation should enable new users to run their own simulations after a very short time, and to contribute new features soon thereafter.

Here, we present the current version of the ESPResSo-package, which as of this writing continues to be expanded with new features and capabilities undergoing thorough testing before officially announced. Downloads, updates and more documentation can be found on the webpage http://www.espresso.mpg.de/. The distribution of the source code adheres to the open source standards under the GPL License [32]. By this we hope to ignite the further development of our code into a valuable research tool for the soft matter community, beyond the already participating groups, scientific institutes, and researchers at industrial companies.

We start the description of ESPResSo with a general outline of the design ideas that guided the development. The next section gives an overview of the capabilities and algorithms included, followed by a discussion of some implementation details. After this, we present benchmarks for sample systems and compare the speed to other simulation programs in Section 6. The paper is rounded off by conclusions and an outlook about features that will be incorporated in the near future.

2. Program design

This section lists the main design goals and their implementation in the ESPResSo development. They are given in their order of precedence, since as usual there has to be a certain trade-off between competing design goals. This order is also influenced by the special needs of our work group, where we use coarse-grained descriptions of soft matter systems, mainly bead-spring models. However, it had become evident in the past that applying a program to different scientific projects can only be successful if it is easily possible to adjust it to the specific challenges each new project contains. If achieving this requirement results in some performance loss this will be made up for by the smaller amount of human preparation time needed to initiate the actual simulation.

2.1. Goals and principles

Extensibility: Since we want to use our program for performing basic research, objectives can change, system complexities expand, and often new challenges arise—a simulation package must enable the user to add custom code, otherwise it will only be useful for a very restricted group of people. ESPResSo addresses these concerns by having an entirely modular integration and communication core around which all other routines are structured. This allows the exploration of a wide range of models with a vast variety of properties, and gives at the same time solid support for algorithm development. The extensibility of ESPResSo, its routines and its data structures, has been proven during the incorporation of new algorithmic techniques like the Maggs algorithm (see Section 3.3.3) and the lattice Boltzmann algorithm (see Section 7) which both required changes in the core of the program. This makes us confident that the given program structure is well suited for further not yet foreseen changes and additional features.

Readability: Unless the structure and coding style are kept simple, extensibility would remain a theoretical issue. That is why ESPResSo puts much emphasis on a clear structure and thorough documentation, to ensure that programmers with various degree of programming skills are able to gain access to the different coding layers that form the simulation engine. Naturally, beginners would want to start off with smaller tasks such as implementing a new potential (maybe following the *How To*-Guide in the documentation) before attempting to alter core sections of ESPResSo, since the latter’s complexity naturally requires an increased experience from the user.

Flexibility: ESPResSo's scope of general applicability discourages the usage of highly specialized and optimized functions. Flexibility requires looking at each functionality or data structure from multiple directions to ensure that it is useful to a large variety of problems. Even if most enhancements probably originate from a very concrete application, the aim is to always extend them towards a more universal usage before they are released. Examples for this flexibility are the usage of a very general Lennard-Jones potential which enables the simulation of colloidal particles and the implementation of a particle structure which allows also the treatment of anisotropic particles.

But this is only one part of ESPResSo's flexibility. The employment of Tcl as the steering level opens vast opportunities for the design and course of intended computer experiments. Moreover it opens the opportunity to implement new requirements on a script language level at the same time being still in close linkage to the program core. This has, for example, been practiced for hybrid Monte Carlo/MD simulations. This approach saves a huge amount of human development time, at the expense that the resulting program will be somewhat slower than a highly optimized and specialized one that we however accept for the aforementioned reasons.

Correctness: ESPResSo is not only a multi-user, but more importantly a multi-developer program where the full source code is accessible to everyone. This has the advantage that whoever is a specialist in a certain (algorithmic) field will be implementing the desired functions with expert knowledge. However, every individual contributor to the ESPResSo-project must be enabled to run a thorough check on the added code fragments to ensure that the numerical and physical correctness of the package as a whole remains intact even after the changes were applied.

For this purpose, ESPResSo includes a testsuite of various cases which quickly evaluates if the most important features are still functioning properly; since this check takes only a few minutes, the inhibition threshold for developers to use is low. In addition, there are also more elaborate test scenarios and sample scripts, that contain well known physical problems and systems that were solved some time ago. These will provide a very thorough review of ESPResSo's overall integrity by comparing the computed results to the published reference data; although these tests take longer, they help to avoid programming bugs. Even though these automatic control features already ensure a high degree of reliability of the package, we found it still necessary to check the correctness and consistency of any larger change of the code by hand in a team effort.

Efficiency: Last, but surely not least, the applied algorithms should naturally be the most modern ones available, to ensure a fast execution time of the simulation. For this purpose ESPResSo incorporates state-of-the-art electrostatic routines such as $\mathrm{P^3M}$ or ELC, provides multiple cell systems for choosing a particle storage organization optimized for the investigated systems, and constantly adds new algorithms (e.g., a recent suggestion for treating electrostatics locally [33,34]) providing new features. However, all these attempts must still fulfill the previous design criteria, particularly *readability* and *flexibility*—i.e. no hardcoded assembler loops or platform dependent computation tricks will be implemented, because the achievable speedup through this would most likely render further attempts of enhancing the program much harder, if not impossible.

These wide specifications and conflicting design goals enforced a thorough thinking about the program architecture, data structures and general programming guidelines for the development of ESPResSo. Since ESPResSo is a team project where basic concepts evolve in a discussion process, further adjustments may be implemented, always having feasibility and applicability in mind. In the end, everything will be documented in an understandable and usable way to ensure coherence and compliance for future releases.

2.2. Basic program structure

ESPResSo is organized in two hierarchical program levels, each suited to optimize some of the main targets we had in mind: the steering level for large flexibility, usability and extensibility, the simulation engine level for efficiency, readability and again extensibility. The two program levels are:

Steering level: The steering of ESPResSo is done on a script language level, namely Tcl/Tk [35]. All tasks are implemented as C-functions enhancing the Tcl-interpreter and acting as new Tcl-script commands. This includes input and output of data, setting of particle properties, interactions, and parameters, and performing the integration and analysis of a given system. All commands that interact with the simulation, are built such that they allow both for setting and retrieving information. For example, with the command `part`, which deals with particle properties, one can set the particle coordinates, but one can also retrieve the particle coordinates at later times. A sample simulation script for a Lennard-Jones fluid is given in Appendix A.

But steering a simulation in ESPResSo is much more than setting up system parameters. From the steering level one can interact with the actual simulation in a very flexible way. That means one can change the system properties during the simulation in any desired way, may this be the insertion or deletion of particles, a change of interaction parameters, a switch from one to another thermodynamic ensemble, a change of parameters for the integrator or the thermostat—just to name a few possibilities. This allows, e.g., for simulated annealing by modulating the temperature, or to perform a thermodynamic integration by gradually changing the Hamiltonian of the system.

However, also in the equilibration of most systems, this feature is frequently used: For the simulation of dense, charged polymer melts, as another example, one could start the equilibration without electrostatics, add polymers and some neutral spheres (the latter to become counterions later on) at the correct density; temporarily capping the excluded volume interactions would then allow to carefully push overlapping particles apart, always supervised by the Tcl-script which could use analysis routines to monitor the minimum distance of all particles until they are separated. Afterwards, the spatial positions of the monomers could be fixed before the electrostatic interactions are activated, to prevent locally unbalanced charges to rupture the polymer's bonds, allowing the counterions to equilibrate. Releasing the chains is then the last step towards the physical system setup of a poor solvent polyelectrolyte melt.

All steps described above are typically part of the Tcl-script that steers the simulation. Even more complex tasks can easily be formulated with the Tcl language. It is also possible to conduct a wide range of data analysis during runtime of the simulation, which is not only convenient but also allows to feedback analysis results into the steering of the simulation flow, e.g., to implement cross checks surveying the kinetic temperature in an isothermal ensemble or the instantaneous pressure in (N, p, T), to initiate system configuration swaps, something needed to use *parallel tempering* [36], or to change the coarse graining level of the physical system. This can involve replacing explicit particles with a larger and appropriately heavier center-of-mass representation, and vice versa.

With all these possibilities available, the user is never constrained to a specific system, ensemble, or configuration. One can, in principle, modify all of the initially chosen properties during runtime. Of course, instead of using the slower, but more flexible Tcl interface, the user has always the option of adding features to the engine level of ESPResSo. As already mentioned in Section 2.1 the steering level enables the user to carry out a huge part of the programming work required for solving new problems on the script language level rather than elaborate C-programming. Even though for standard simulations the script can be as simple as the one shown in Appendix A the given possibilities can yield quite complex simulation scripts. These scripts perform not only the steering of the simulation, but also analyze the data on the fly, report regularly the observables of interest, and export graphs of the observables. The steering level also allows to manipulate the simulation from a graphical user interface (Tk) or via input from external programs.

Simulation engine level: In contrast to the steering level, which uses the Tcl-script language for easy usability, the simulation engine has to be efficient and is therefore implemented in ANSI-C. The engine is responsible for storing the particle and interaction data on any number of processors, but it also performs the integration of Newton's equation of motion as well as the necessary calculations of basic quantities like forces, energies and pressures.

The simulation engine code has been organized to be as modular as possible, separating clearly different tasks like particle organization, interaction organization, integration, or force calculation, for example. Some advanced algorithms, especially for the calculation of the electrostatic interactions, require deep knowledge about the particle organization, making complex interfacing necessary. Basic functions can be accessed using well-defined lean interfaces, hiding the details of the complex numerical algorithms. For example, the particle data can always be accessed through a defined set of functions and variables, although internally very different particle organization schemes can be applied (see Section 4).

Another example is the implementation of new potentials, which is possible without touching the integrator code, and in turn the integrator can be modified without touching the potential implementations. In this way it is much easier to implement a new type of interaction, a new integrator, or a new thermostat. It is also much easier for a beginner to understand what the program is doing during the simulation. This is important in light of our belief that it is not useful to perform scientific research with a simulation tool that appears mainly as a "black-box" to the scientist in charge.

Besides Tcl/Tk, ESPResSo relies on another Open Source package, namely the FFTW [37] in the P^3M method for the electrostatic interaction. For the parallelization MPI routines are used; on platforms running Linux and Darwin ESPResSo uses the LAM/MPI [38] implementation, offering the option to employ other packages such as MPICH [39] as well.

3. Capability and algorithms

In this section we try to give a short overview of the algorithms and features already implemented in ESPResSo as of this writing. These can range from different integrators over particle interactions up to data analysis routines, for e.g. the radius of gyration. During the start of our program most of the implemented algorithms had been devoted to (charged) polymers, rod-like objects, or membranes [30], while today ESPResSo is not limited to these applications anymore. Many features have been added in the meantime, including the possibility to perform atomistic simulations and to employ various coarse graining techniques. Since ESPResSo is constantly being extended, we keep an up-to-date listing in the Release Notes-file of the distribution and on our webpage (http://www.espresso.mpg.de).

3.1. System setup

For a successful computer simulation a thorough preparation of the system to be investigated is the essential first task. The Tcl-script level of ESPResSo allows for an easy and variable setup of the starting configurations together with the simulation parameters, as well as choosing the desired algorithms. From placing particles and charges, defining potentials and interactions, assigning bonds and constraints, applying external forces and customized boundary conditions, to reading in existing configurations in ESPResSo's native blockfiles (see Section 3.5) and other formats, everything can be easily executed using few simple commands within the Tcl-script. It is also possible to rely on pre-defined setup routines and sample topologies such as polymer chains, networks, fullerenes, counterion- and salt-distributions which are provided as shortcut-like conveniences of commonly employed systems, and the user may adjust to any custom needs arising.

Putting particles at random positions in the simulation box, for a simple Lennard-Jones fluid, for example, can be done directly by a single loop:

```
set box_x [lindex [setmd box_l] 0]
set box_y [lindex [setmd box_l] 1]
set box_z [lindex [setmd box_l] 2]
for {set i 0} { $i < $n_part } {incr i} {
    part $i pos [expr $box_x*[t_random]] \
        [expr $box_y*[t_random]] \
        [expr $box_z*[t_random]] type 0
}
```

An example of a pre-defined setup routine is the `polymer` command, which can be used, e.g., to setup a bead spring model of a polymer melt. A melt consisting of 100 chains carrying 200 monomers each [40], can be set up by the command line

```
polymer 100 200 1.0 mode SAW
```

It fills the simulation box with the corresponding number of self-avoiding walks (SAW) of step size 1.0σ (the bond length), once the excluded volume interactions and the bond potentials have been defined.

If one needs to constrain single particles, `part... fix` fixes their spatial coordinates, for example, if one wants to investigate the force acting on them as a function of the (imposed) distance, or if one wants to model immobile surfaces.

The user's influence is not limited to geometrical or topological issues. Besides specifying the particles' interactions, choosing thermostats, or the ensemble (e.g., constant volume or constant pressure), all other aspects of the simulation remain fully customizable in the course of the whole simulation. All parameters may be freely adjusted depending on whatever criterion chosen. This is useful in system equilibration, where interaction can be capped temporarily, or during the integration, e.g., to perform a simulated annealing, or to add/remove/manipulate particles for grand canonical ensembles or multiscale approaches.

3.2. *Integrators and thermostats*

ESPResSo uses as default the *Velocity Verlet Molecular Dynamics* integration scheme [41,42] for rotationally invariant particles, which is robust enough for most applications. If a simulation incorporates particles with rotationally degrees of freedom like dipoles or cigar shaped Lennard-Jones particles, the rotational degrees of freedom are represented by quaternions, and an extension of the velocity Verlet algorithm for rigid body motion is used [43]. Mass and inertia moments are fully implemented, but may be replaced by the default reduced mass $m^* = 1$ for performance reasons if no distinction of different masses is desired. For the simulation of dynamic experiments like a simple shear flow, ESPResSo uses *Non Equilibrium Molecular Dynamics* (NEMD) [44,45]. In addition to the intrinsic (N, V, E)-ensemble, the velocity Verlet integration scheme can be combined with one of the implemented thermostats to obtain integrators for the (N, V, T)-ensemble. At the moment these include a *Langevin thermostat*, a *Berendsen thermostat* [46] and a *dissipative particle dynamics thermostat* (DPD) [45,47]. The type and the temperature of the thermostat can be changed during the simulation on the script level, allowing, e.g., simulated annealing. The DPD thermostat implements the friction and noise as a particle pair force rather than a global one, and thus maintains momentum conservation, which is important for yielding correct hydrodynamic interactions for equilibrium and non-equilibrium simulations [45].

Apart from (N, V, E)- and (N, V, T)-ensembles, ESPResSo is also able to keep the pressure constant during the particle propagation, i.e. to simulate the (N, p, T)*-ensemble* [48]. This is implemented by introducing an artificial piston mass which acts on the simulation box and rescales its dimensions isotropically. With the Langevin-type equations of motion for this new degree of freedom one arrives at a stochastic MD integration scheme, enhancing the MD approach by Andersen, Nosé, and Hoover with stochastic dynamics (SD), and follows the momentum of the piston such that its derivative corresponds to the difference between the currently measured "instantaneous" pressure and the given external one. There is also a corresponding isotropic thermostat available which adds friction and noise. In addition to the implementation given in [48], our algorithm allows for using a constant tension ensemble as well, where only the pressure in one or two dimensions is considered, the rescaling adjusted to respond accordingly; this is useful when studying, e.g., membranes under tension [49].

3.3. *Force calculation strategies*

ESPResSo distinguishes three kinds of forces: long ranged forces, short ranged forces, and bonded interactions. Short ranged interactions have to be calculated only for particles within a small interaction range. To avoid unnecessary operations, information from the cell structure can be used for eliminating particle pairs that are far away from each other. Therefore, each cell structure comes with an optimized force calculation routine for the short ranged forces. For example, when the particles are sorted by the domain decomposition cell structure (see Section 4.1), only particles in adjacent cells can interact. This reduces drastically the number of particle pairs that have to be considered. Additionally ESPResSo keeps a record of the particles interacting currently in so-called *Verlet lists*. These reduce the computational work by another factor of around 8, since the lists have to be updated only every few time steps. Currently, only short ranged pair-interactions are implemented, but extending ESPResSo to three or more body forces should be easily possible.

The bonded interactions are treated together with the short ranged interactions. There is no limitation on the number of bond partners, i.e. many-body bonded interactions are already implemented in ESPResSo. The efficient treatment of long range forces is only possible using highly sophisticated algorithms. Those will be discussed later in a separate section.

3.3.1. Short ranged non-bonded potentials

ESPResSo features the following non-bonded short ranged potentials:

- Lennard-Jones potential. In our implementation the cut-off radius of this potential can be shifted to obtain a purely repulsive interaction (so-called WCA potential [50]), or we can have an additional hard-core offset radius. The Lennard-Jones can be capped during the warm-up phase of the simulation, such that even overlapping particles feel only a finite repulsive force. This allows for an easy random placing of particles in an initial configuration.
- Combined Lennard-Jones-cosine potential. If the Lennard-Jones potential is not cut-off in its minimum, e.g., if one needs the attractive tail, the forces are discontinuous. This potential adds a monotonous part of the cosine function to ensure that the force are continuous everywhere [51].
- Buckingham potential [52]. This is another potential mimicking a van-der-Waals type interaction between particles. This is a widely used force field in atomistic simulations.
- Gay–Berne potential [53]. This is essentially the anisotropic version of the Lennard-Jones potential, i.e. for cigar shaped objects. It has to be used in conjunction with an integrator for rigid bodies.
- Debye–Hückel potential [54]. This potential describes a screened Coulomb interaction, which is short ranged. With a screening length of infinity one gets the pure Coulomb potential. Note, however, that in this special case the integration loop is of order $\mathcal{O}(N^2)$ in the number of particles N.
- Tabulated potentials. ESPResSo can handle any number of tabulated potentials, where the force and the energy is described as a piecewise linear function of the distance with finite support.

There are special model systems such as phantom networks, in which there are no nonbonded interactions between particles of the same chain. This behavior can be implemented using different particle types. This is extremely inefficient if several thousand chains are included. For this case, ESPResSo offers exclusions, which allow to specify particle pairs, for which the nonbonded interactions are not calculated, for example, interactions along the chain backbone. These exclusions can be set manually. In standard atomistic simulations one typically excludes the interactions of each particle with its neighbors within the molecule, which can be done automatically by the `part auto_exclude` command.

3.3.2. Bonded potentials

The bonded potentials are:

- *Pair forces*: Two particles can be bound by FENE (finite extensible nonlinear elastic) [51] and harmonic bonds. As an alternative to the exclusion mechanism, ESPResSo features a two-particle Lennard-Jones subtraction potential, which is simply a bonded version of the Lennard-Jones potential with inverted sign. The exclusion code adds some additional overhead to the nonbonded force calculation, since one needs to check for each pair of particles, whether they interact. If only a few interactions have to be left out, it may be more efficient to calculate the nonbonded interactions as usual, and to subtract the interaction of non-wanted particle pairs.
- *Three-body forces*: Three particles can be bound by a bond-angle potential. ESPResSo implements three different kinds of bond-angle potentials. First, a harmonic potential around the optimal angle ϕ_0, i.e. $1/2k(\phi - \phi_0)^2$, then a cosine potential $k(1 - \cos(\phi - \phi_0))$, and finally a harmonic cosine potential $(1/2)k(\cos(\phi) - \cos(\phi_0))^2$.
- *Four body forces*: Four particles can be bound by a dihedral angle potential. The implemented form is $k(1 + \delta\cos(n\theta))$, where θ is the dihedral angle, $n = 1, 2, \ldots, 6$ is the multiplicity and $\delta = \pm 1$ is a phase factor. With this potential one can also mimic force fields which use $\sum_i k_i \cos(\theta)^i$ as the functional form for the dihedral interactions.

In addition to the potentials above, bonded potentials can be given in a tabulated form, where the energy and the force are given as piecewise linear functions of the distance (two particles), the bond angle (three particles), or the dihedral angle (four particles).

3.3.3. Long range potentials

While the implementation of the short ranged potential normally consists only of two short routines, namely the force and energy calculations, the treatment of long ranged interactions requires much more care due to the large number of possible interaction partners. The algorithms for long range interactions are highly sophisticated [5] and normally have their own parallelization strategies. This enormous effort is justified by the fact that a simulation with electrostatic interactions might spend more than two thirds of the computation time only for the calculation of the long range interactions. At the moment ESPResSo only handles Coulomb interactions since this is the dominant force for soft matter problems, but fast algorithms for dipolar interactions are being currently implemented.

For any finite geometry without periodic boundary conditions we simply use a loop over all N charged particles which is of $\mathcal{O}(N^2)$. For 3D periodic boundary conditions the standard Ewald method has in its optimal implementation still a complexity of $\mathcal{O}(N^{3/2})$. To overcome this limitation we have implemented the P^3M algorithm of Hockney and Eastwood [55] for 3D periodic boundary conditions in a version as described in detail in Ref. [8]. This method is the optimal choice for the Fast Fourier transform based "particle mesh Ewald" algorithms, since it uses the pair-force-error-optimized lattice Green's function, and by construction is superior to other variants [56,57]. The computational effort scales with $\mathcal{O}(N \log N)$. An additional strength is that robust error estimates exists [9]. Our implementation handles arbitrary dielectric constants at the boundary. For the Fourier transformation we use the FFTW which exists for various platforms [37]. Our parallel implementation employs MPI and the one-dimensional FFTW. The implementation of ESPResSo is more flexible than the parallel multidimensional FFT employed in the FFTW, since it does not limit the maximal number of usable processors to be equal to the mesh size.

For systems with periodicity only in 2 or 1 dimensions, ESPResSo features the MMM2D [10,58] and MMM1D algorithms [59]. They are based on a convergence factor approach to evaluate the Coulomb sum. These algorithms are implemented for arbitrary box shapes, but have scalings of $\mathcal{O}(N^{5/3})$ (2D) and $\mathcal{O}(N^2)$ (1D). They also feature robust error estimates, and one of their advantages is that high numerical precision is not computationally costly. For a more complete discussion see [10,58,60,61]. They are faster and more robust than the 1D and 2D standard Ewald methods, or Lekner sums [62].

The scaling of MMM2D of $\mathcal{O}(N^{5/3})$ normally allows for not more than 1000 particles to be treated in a simulation. For higher numbers of particles, or lower precision, as it is needed for dense electrostatically interacting systems as in biomolecular applications, ESPResSo features the ELC method, which stands for electrostatic layer correction [11,12,61]. Here one uses an algorithm for fully periodic boundary conditions with its favorable scaling (in ESPResSo currently only P^3M) for a 3D system, where only two dimensions are periodically replicated. The summation order is made slabwise by a simple change in the dipole term [63]. The unwanted interactions of particles with their images in the non-periodic dimension can be analytically computed by means of the ELC term, and then are subsequently subtracted [11,12]. This term scales linearly with the number of charges and therefore keeps the favorable scaling of the P^3M method.

All our above described methods can be preset to a user defined accuracy, and will be optimized to the underlying hardware (at this set error) by means of a tuning routine. The ability of choosing the accuracy and the tuning facility for all those algorithms is unique to ESPResSo and is, as far as we are aware, absent in other packages.

In addition to the above described methods ESPResSo possesses a local electrostatic solver due to Maggs [33,64] in a version of Pasichnyk and Dünweg [34] that was developed for MD simulations. The essential philosophy of this approach is to view electrostatics as the quasi-static limit of a dynamic field theory, in this case Maxwell electrodynamics, and to explicitly simulate the dynamics of the field degrees of freedom. The speed of light c appears as an adjustable parameter and realistic dynamics occurs for particle velocities $v \ll c$. The Maxwell equations are discretized on a simple-cubic lattice, with scalars on the sites, electric fields on the links, and magnetic fields on the plaquettes. The method is rather simple to implement and to parallelize, and appears to be quite competitive with respect to efficiency. In principle, it should allow for a locally varying dielectric constant [64], also this feature has not been implemented.

3.4. Constraints and external forces

For a non-bulk simulation the particles normally have to be constrained to some finite region of the space. In ESPResSo these constraints are implemented as soft immobile objects, i.e. they interact with the particles through one of the short-ranged potentials. ESPResSo features basic constraints such as walls, cylinders or spheres, which can be arbitrarily combined to form the desired simulation region. Walls and cylinders with infinite extension, i.e. in periodic boundary conditions, can carry a charge for simulations of charged surfaces. Besides this, more exotic constraints such as a two-dimensional array of spherical cavities interconnected by circular holes, are available.

Other type of constraints are positional constraints and external forces. While the former type of constraint simply makes a particle immobile in some or all of the spatial coordinates, allowing, e.g., pulling at a molecule fixed to a surface, the latter exerts a constant force on specific particles as in an external field. For example, the potential of mean force between two macromolecules can be determined by fixing their center-of-mass distance. A `comfixed`-constraint fulfills this task, which allows to explore the configurational phase space as a function of the center-of-mass separation.

```
{variable
        {box_l 316.950192362 316.950192362 316.950192362}
        {periodicity 1 1 1}
        {time 5200}
} {interactions
        {0 FENE 7.0 2.0}
        {0 0 lennard-jones 1.75 1.0 2.5 0.004079222784 0.0 0.0 }
} {integrate
        { set nvt }
} {thermostat
        { langevin 1.0 1.0 }
} {particles {id pos type q}
        {0 -0.933410431074 -4.396367828 4.07839266194 0 1.0}
        {1 80.6308288023 73.5960975815 74.5619693121 0 1.0}
...
        {3583 28.8582652959 75.2256394353 412.044956716 2 -1.0}
} {bonds
        {0 { } }                    ...
        {206 { {0 205} {0 1} } }    ...
} {tclvariable {V1 31840000.0} }
```

Fig. 1. Example sketch of an ESPResSo-compatible blockfile (excerpt), displaying some of the different categories as described in the text. The *variable* block contains informations on the simulation box (box length and boundary conditions) and the current simulation time in the `variable` block. The *interactions* block describes a bonded FENE- and a nonbonded Lennard-Jones-potential. The *integrate* and *thermostat* blocks contain integrator settings for integrating an (N, V, T)-ensemble with a Langevin thermostat at temperature $T = 1\epsilon$ and friction $\Gamma = \tau^{-1}$). Afterwards, unique identifier, 3D position, type number, and charge are given for all particles in the system. The *bonds* block shows that no bond is stored with the zeroth particle, while the 206th is connected to particle 205 and particle 1). Finally, the value of a user-defined Tcl-variable `V1` is included, which could for example represent a parameter describing the system or an observable.

3.5. File input/output

Although Tcl provides built-in support for reading and writing (text-)files, ESPResSo introduces a unified file format for saving and (automatically) restoring its computer simulation data to allow easy exchange between users. Based on the structure of Tcl-lists, this *blockfile format* subdivides all informations on the current state of the program into categories (*blocks*). An example of an ESPResSo compatible blockfile is shown in Fig. 1.

For each block type routines exist to write the block to a blockfile and to read its data back into the running simulation. ESPResSo handles many block types by default, such as *variable* (contains global parameters such as simulation box length, skin, processor node grid, cell grid, chosen periodicity), *tcl-variable* (contains Tcl-variables, such as important constants in the simulation script), *interactions* (contains a list of all the employed bonded, non-bonded, and electrostatic potentials with their parameters), *integrator*, *thermostat*, *particle* (contains informations such as positions, velocities, forces and charges), *bonds* (contains the bonding topology of the system), *random* (contains the state of the random number generators) and *configs* (contains particle informations of past timesteps for online-analysis of their trajectories).

In addition to these standard blocks, ESPResSo allows to add arbitrary blocks written in Tcl. Also the default block types are implemented as Tcl routines. To add a new block type, one has to define only two Tcl procedures, one which is used to write the block data and one to parse it. This feature can be used to conveniently write and parse observables. Note that the implementation of these procedures is easy, since most ESPResSo commands are able to output their parameters in a form that can serve as their own input.

The amount of informations to be included may be chosen entirely by the user. Some shortcut commands exist as well, i.e. to create a simulation system snapshot which includes everything necessary for starting or resuming an integration. The output will be saved in plain text (ASCII) format ensuring great flexibility, portability, and (more importantly) platform independence compared to system dependent binary formats; the unavoidable size disadvantage is easily compensated when compressing the resulting files (which can be done on-the-fly) using external programs such as gzip.

ESPResSo supports *checkpoints*, that can be created regularly during the simul-ation. These can be used to restart the system in a well defined state, or for later off-line analysis of particle trajectories. For this matter, invocation of

```
checkpoint_set custom_name.$i.gz
```

from the Tcl-script will create a blockfile `custom_name.$i.gz` which will be compressed automatically, and which stores all available information about the currently executed simulation. This file represents the *i*th entry to the list of checkpoints. Its filename will be appended to `custom_name.chk`, allowing to keep track of the series of checkpoints built so far. Whenever it is necessary or desired to resume the program,

```
checkpoint_read custom_name.chk
```

will successively rebuild the stored trajectory, continuing the simulation *exactly* at the data status of the last checkpoint.

In addition to reading its own data format, ESPResSo is also able to read or write other file formats, such as pdb/psf-files which are used by visualization software such as VMD [65] and file formats of simulation packages previously used in our work group. There are is an ongoing project to include import and export of GROMOS and GROMACS files. Due to the use of the Tcl-script language, adding new text file formats is particularly easy, which is often necessary if data from self-written simulation codes should be imported to ESPResSo.

3.6. Data analysis

ESPResSo does not only supply the simulation kernel, but provides also tools for analyzing the data. This can be done both, after the end of the simulation, and on-the-fly, i.e. while ESPResSo is still running. This is convenient since some simple plausibility checks can be performed, e.g., for following energy conservation or the temporal evolution of selected observables, allowing to abort or adapt the simulation accordingly. Typical derived observables range from the forces exerted on specific particles, the energy, particle distributions, or typical properties of chain molecules. The pressure can be calculated both tensorial and isotropic.

Radial distribution functions can be computed for specific particle types, allowing for, e.g., the calculation of the distribution of counterions around a macromolecule. Implemented observables for chain molecules are the average end-to-end distance R_E, the radius of gyration R_G, the hydrodynamic radius R_H, the average bond length $\langle b \rangle$, the internal distance distribution, and the mean square displacements g_1 of the particles, the mean square displacements g_2 of chain particles relative to the chain's center of mass, and the center of mass displacements g_3. For comparison with scattering experiments, *structure factors*, both as spatially averaged single-chain form factors for polymers, as well as total structure factors for selected particle species or the entire system, can be calculated. For membrane systems, ESPResSo can perform a fluctuation mode analysis [66].

Most of the data analysis does not only average over the current sample, but is also able to optionally average over an entire series of past timesteps stored in memory. For this feature, ESPResSo allows to keep specific configurations loaded, building up an entire trajectory for online analysis; of course, this trajectory can also be saved to disk for later re-usage employing the blockfile format.

For a simple error analysis of the averaged observables, the standard deviation of the measured values is computed. The derivation of more abstract error estimates is supported by a simple linear regression, or, for correlated observables, the *uwerr* command, which calculates the errors of values derived by, in general, nonlinear functions from observables with known correlated errors [67].

3.7. Additional features

In addition to the built-in analysis procedures, there is a fast growing number of Tcl procedures for mathematical treatments, analysis, and general data manipulation. The extensibility of ESPResSo also brought up some more unusual features: ESPResSo can make use of the Tcl extension Tk, which allows for the creation of graphical interfaces. This is useful primarily for writing scripts for visual simulation demonstrations. To this aim, ESPResSo also has an IMD interface that can send configurations during the simulation to the visualization package VMD [65] which can provide animated movies of the simulation trajectories. ESPResSo can automatically generate postscript graphics of graphs using *gnuplot* for direct plotting of any arbitrary combination and functional expressions of measured observables.

3.8. Test-suite

Due to the scope and sheer size of ESPResSo it is an important task on its own to ensure that—despite the easy access of all users to all parts of the program—it is possible to always ensure the physical correctness of the entire package, its reliability, and backwards compatibility. To this aim, each of the major features in ESPResSo (e.g., each potential, each integrator, the thermostats, common data analysis routines, specialties such as constraints, partial periodic boundary conditions, external forces, etc.) have their own little test-case which quickly compares the output of the current version of the program to an earlier reference state, thereby detecting any deviations (maybe inadvertently) introduced in between.

Before submitting modifications to the CVS-repository for future releases of ESPResSo, each developer is required to ensure an error- and warning-free compilation on *all* the officially supported hardware platforms, checking for each architecture whether the test-suite runs through without any problems. The test-suite is build into the project's makefile for the compilation; hence, it can be executed with a single command. By construction all the tests only add up to a few minutes of computation time to encourage developers to use it as often as possible.

In addition, larger checks testing physical properties of "real" systems provide simple means to also trace more subtle bugs only occurring for specific scenarios or unique combinations of features or circumstances. Being based on previous research projects or "classic" simulation scenarios (e.g., the Kremer–Grest study of a polymer melt [40]), these may take considerable longer time to complete. The comparison of the output of the current program version to the published data, however, allows for an excellent test of the overall correctness of ESPResSo. Done on a regular basis, all these verification scenarios are executed and need to be fulfilled when releasing a new version of the code. In this way we hope to keep the number of newly created errors in updated releases bound to a minimum, and to enhance stability of our program package.

4. Implementation details

After having described the overall structure, capabilities, and included algorithms of ESPResSo we will present in this section some details of the implementation, especially of the particle data organization.

4.1. Data structures for parallel computation

The data ESPResSo needs during the integration are mainly particles, interactions, and constraints. While the number of interactions and constraints is normally small, and simple lists are efficient enough for their storage, the particle data needs some more elaborate organization. A particle itself is represented by a structure consisting of several substructures, which in turn represent basic physical properties such as position, velocity, force, mass (if enabled), or charge. The particles are organized in one or more particle lists on each node, called *cells*. The cells themselves are again arranged by several possible systems, which are called *cell systems* in the following. A cell system defines how the particles are stored in ESPResSo, i.e. how they are distributed onto the processor nodes and how they are organized on each of them. ESPResSo currently supports three cell systems, namely an N^2-model, a layered model, and a domain decomposition model. For most simulations the last model is used, which is shown schematically in Fig. 2. The cell models will be discussed in more detail in the following.

Technically, a cell is organized as a dynamically growing array, not as a list. This ensures that the data of all particles in a cell is stored contiguously in the memory. The particle data is accessed transparently through a set of methods common to all cell systems. These allocate the cells, add new particles, retrieve particle information and are responsible for communicating the particle data between the nodes. Therefore, most parts of the code can access the particle data safely without direct knowledge of the currently used cell system. Only the force, energy and pressure loops are implemented separately for each cell model, since the calculation can benefit in different ways from the particle organization. For example, the computation of the pair forces can be implemented in linear computation time using the domain decomposition method, which will be explained now.

The *domain decomposition cell system* is based on the *link-cell algorithm* [68–70], which will now be briefly reviewed. Many pairwise interactions, such as the Lennard-Jones interaction, are short ranged, i.e. their value is small enough to be neglected at a distance much smaller than the size of the simulated system. Therefore the interactions have to be calculated only with particles close by, but since one has to check the particle distance for every pair of particles, this would still result in an algorithm with a computation time scaling of $\mathcal{O}(N^2)$, i.e. the number of necessary operations grows quadratically with the number of particles. A more clever way is to set up so called *link cells* at the beginning of the simulation. These cells cover the entire simulation box and contain links to all particles in their spatial domain. Their size is chosen slightly larger than the maximal interaction range. In the force calculation only interactions between particles in adjacent cells have to be calculated, so that one only has to run through

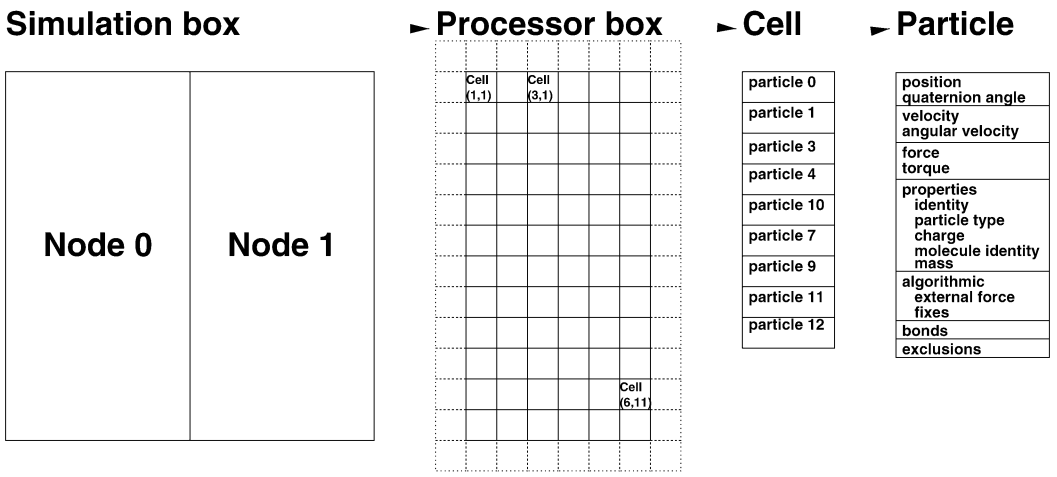

Fig. 2. Schematic description of the particle data organization using the domain decomposition cell system within ESPResSo. The simulation box is split up into equally sized regions assigned to a single processor each. The region assigned to a processor is then split up spatially into cells (solid lines). Around the cells a shell of additional cells is wrapped that will contain data of the particles from neighboring processors (dashed lines). The cells are simple arrays of particles, which finally consist of several substructures containing the position, force and other particle data. Depending on the type of simulations, the detailed contents of the substructures can vary.

the 27 neighbors for each cell (in three dimensions). At constant density and interaction ranges, the average number of particles in a cell is constant and therefore the overall algorithm has order N (since we run once through all cells and therefore all particles).

The force calculation for this system can easily be parallelized as well: First the simulation box is split up equally into as many smaller boxes as processors are available, and each of these boxes is assigned to a processor. The particles are then assigned to the processor that is responsible for the box they are located in. Now these processor boxes are divided up into cells. Each node calculates the interactions for all particles assigned to it. But to do this, the particles located in the cells adjacent to the processor box are needed, too. This shell of cells around a processor box is called the *ghost shell*, and the particles in the ghost shell are called *ghost particles* [25]. The ghost particles have to be communicated between the nodes, while all other particles have to be known only on their respective node.

After propagating the system by one timestep, some particles may have moved out of their cell, which would require to update the cell lists. In a multiprocessor simulation, that could even mean shifting around particles from processor node to processor node. To reduce the number of particle reorganizations, one can exploit that the cell size is slightly larger than the maximal interaction range. The difference between cell size and maximal interaction range is called the *skin* distance r_{skin}. As long as no particle has moved further than $r_{\text{skin}}/2$, the distance of two particles did not increase by more than r_{skin}, so that adjacent cells still contain all interacting particle pairs, and the link cells do not have to be updated. Therefore, with each particle one stores the position where the last sorting process took place, and checks whether this position is further away from the current position than $r_{\text{skin}}/2$. If this happens, the link cells are rebuild. The optimal value of r_{skin} is hardware dependent as it reflects the tradeoff between having to treat more particle pairs due to the increased cells size versus having to update the link cells. The skin can be as large as 20–40% of the maximal interaction range, and normally the lists are updated about every 20–40 timesteps using ESPResSo on a Linux PC. The skin value has to be determined manually by, e.g., running several integrations at different skin values, since the optimal value depends heavily on the underlying hardware, especially on the performance of the inter-node communication. To safely cover a wide range of simulation conditions, in contrast to many other programs, ESPResSo checks after each integration step if the Verlet list has to be updated.

The domain decomposition cell system of ESPResSo implements the link cell algorithm. But instead of just having links to the particles in the cells, the cells contain the particles themselves in an array. For an example let us assume that the simulation box has size $20 \times 20 \times 20$ and that we assign 2 processors to the simulation. Then each processor is responsible for the particles inside a $10 \times 20 \times 20$ box. If the maximal interaction range is 1.2, the minimal possible cell size is 1.25 for 8 cells along the first coordinate, allowing for a small skin of 0.05. If one chooses only 6 boxes in the first coordinate, the skin depth increases to 0.467. In this example we assume that the number of cells in the first coordinate was chosen to be 6 and that the cells are cubic. ESPResSo would then organize the cells on each node in a $6 \times 12 \times 12$ cell grid embedded at the center of a $8 \times 14 \times 14$ grid. The additional cells around the cells containing the particles represent the ghost shell in which the information of the ghost particles from the neighboring nodes is stored. Therefore the particle information stored on each node resides in 1568 particle lists of which 864 cells contain particles assigned to the node, the rest contain information of ghost particles.

If one particle has moved further than the $r_{\text{skin}}/2$, all particles have to be sorted into their correct cells, as discussed above. In the case of ESPResSo, this does not only mean to change the pointers in the cells, but the particle data has to be physically moved to another particle list. This creates obviously considerably more overhead than just changing link pointers. On the other hand, one avoids a lot of indirect particle accesses through the link cell pointers, and the particle data is accessed in larger contiguous blocks. This is advantageous on modern computers, since random memory access is quite slow. While a contiguous block of memory can be read at a rate of more than 1 GB/s or 250 words per microsecond from double data rate memory at 200 MHz, one can read less than 50 words per microsecond from random memory positions, reducing the performance to 200 MB/s. In contrast to this, an AMD Opteron processor can multiply 500 double precision floating point numbers per microsecond. To bypass the memory bottleneck, modern processors have small amounts of directly attached fast memory, the caches. These allow to access the data at processor clock speed, which corresponds to more than 700 words per microsecond. However, these can normally only contain the data of a few 1000 particles. The approach of ESPResSo is designed to minimize the effects of the memory bottleneck even if the processor caches are not sufficient.

The optimal skin size depends on the underlying hardware. If the memory system has a high throughput compared to the floating point performance, it is favorable to use small skins. This minimizes the number of possible interaction partners, but requires frequent particle re-sortings. For low memory bandwidth systems, large skins are more suitable, as they reduce the frequency of particle re-sortings at the expense of an increased number of interaction partners. This effect will be demonstrated below in the benchmarks section. To find the optimal size, a simple tuning script is provided with ESPResSo. Due to its different memory organization, the optimal skin values of ESPResSo are slightly larger than the values typically used in other programs, which simply reflects that the particle reorganization is more costly, while the interaction calculation is less costly.

Another side effect of this data organization is a slightly more readable code. The standard link cell algorithm requires a lot of indirect accesses to the particles through the cell pointers in the particle re-sorting procedures, which are not needed in ESPResSo. On the other hand the transfer of a particle from one cell to another is not more complicated than the update of a link address in the code, as this is handled by a separate subroutine.

The other two cell systems, namely the N^2 cell system and the layered cell system, are not as efficient as the domain decomposition and only have to be used with certain potentials. The N^2 cell system will calculate the interactions for all particle pairs. This is necessary, e.g., for MMM1D, a method for the calculation of the electrostatic interaction in one-dimensionally periodic systems, or for the calculation of the electrostatic interaction with no periodic boundary conditions, e.g., like in a simple cell model that possesses hard or open boundaries. Since all interactions have to be calculated anyway, a domain decomposition is unnecessary. Instead of this, the particles are load balanced at the beginning of the simulation, i.e. the particle number does not differ by more than one from the average particle number on each node. Once this load balancing is achieved, the particles are not re-sorted again. This method requires only one particle list per node.

The layered cell system is very special and is a combination of the domain decomposition and the N^2-method. The system is split up into cells or layers only along the z-coordinate. Interactions are treated with all particles in the adjacent layers. This cell system probably only makes sense in combination with MMM2D, a method for the calculation of the electrostatic interaction in two-dimensionally periodic systems.

The concept of cell systems allow for quite different data organizations within ESPResSo, as is needed for some state of the art algorithms. The way that ESPResSo stores the particle data is to our knowledge unique and highly efficient. In addition, the program code is somewhat easier to read. It is easy to add new cell systems if a new algorithm requires a different particle organization. Therefore, this data model fits optimally to our main goals: readable code, state of the art algorithms, efficiency, and of course, extensibility.

4.2. Topology information in ESPResSo

For the storage of the topology information, ESPResSo tries to provide data structures that are optimized for specific algorithmic tasks. Those can be quite different, e.g., integration versus data analysis, and consequently ESPResSo has different ways of storing this information.

For the integrator it is sufficient to store the topology information locally at each particle. This is realized with a bond list at each particle, containing the type of a bond and the identification of the bond partners. In this way the user is free in specifying any topology on the script level and the topology information is automatically distributed to the node where this information is required for the integration. Usually this is sufficient for the integrator making any additional topological concepts like molecules, polymer chains or residues in a protein unnecessary (just think about what the term molecule means in a random network?). Nevertheless, for some applications one might want to distinguish inter- and intramolecular interactions, e.g., in a phantom model [71,72]. For this purpose, each particle has, in addition to the particle type, a molecule identity which then can be linked to the non-bonded interaction potentials.

In contrast to the integrator, for purpose of data analysis the topology is stored in a global way, grouping the particles into molecules. The topology is then stored as a list of molecules each of which consists of a list of particle identities. This is, for example, useful for calculating molecular properties like the radius of gyration, the hydrodynamic radius or the structure factor (form factor).

The three topology data structures are by default independent from each other, and can, for flexibility, be accessed separately from the script. If desired, it is also possible to synchronize them such that they all contain the same information. A summary is given in Fig. 3. All concepts can handle any topology containing n-body interactions. The number of bond partners is just restricted by the types of potentials implemented.

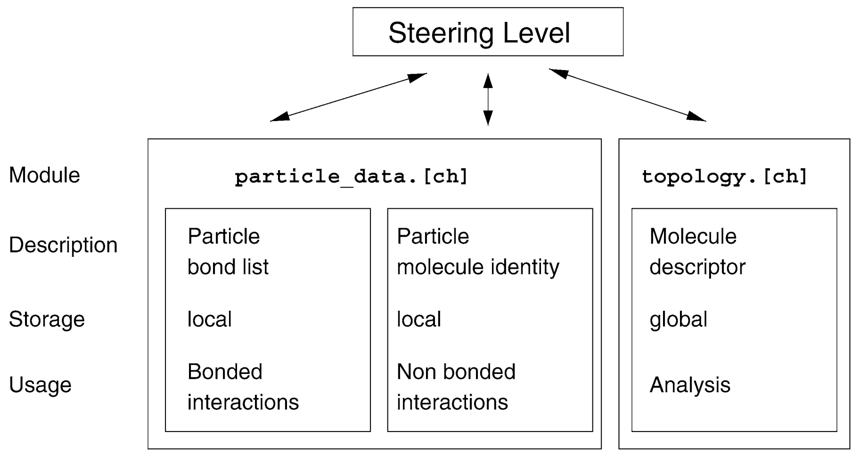

Fig. 3. The three different topology concepts in ESPResSo. Shown are the location in the program package, the type of data storage and their main usage.

4.3. Internal program flow

The Tcl script is interpreted by an Tcl interpreter on exactly one of the nodes, called the master node or node 0 in the following. All other nodes are called slaves, and just wait for commands from the master node. The Tcl interpreter will call C-procedures on the master node, that by convention have the same name as the corresponding Tcl commands, i.e. `setmd` will call the C-procedure "`setmd()`", with the variable name and value as (string) parameters. The C-procedure then parses the input, and performs the operations necessary for the command, in this case it sets the value of a global variable. But this is not all—in a multiprocessor environment the change of one parameter has to be communicated to all nodes. Therefore, the master node issues a command to the slaves requesting to change the value of the corresponding variable, too. Since the number and order of commands issued by the master node is not known at compile time, the communication during the script execution is asynchronous.

Another important point is that ESPResSo allows changing parameters at any time, even during the simulation run, which requires some care to ensure consistency. Changing the processor grid, for example, might change the association of the particles to the different nodes, and therefore requires a reorganization of the internal particle structures. If the Bjerrum length is changed, the currently used method for the calculation of the electrostatic interaction has to be reinitialized. Moreover, if this happens during the simulation, the forces stored in the particles for the current configuration are invalid and have to be recalculated. Because of the large number of algorithms implemented in ESPResSo, the dependencies are actually even more complex. They are resolved by handler procedures, for example, `''on_parameter_change()''`, which is called, whenever a parameter changes, or `''on_coulomb_change()''`, which is called every time a parameter of an electrostatic method is changed (inside `''on_parameter_change()''`). To keep the number of these handlers small, they are written in a quite general fashion and often reinitialize more parts of the code than would be strictly necessary. This small drawback in computational speed is more than compensated by the robustness of the code that this procedure produces. Parameter changes are propagated safely through the code, which allows for a greater flexibility at the script level.

One of the asynchronous commands starts the propagation of the system in time, the integration. During the integration, ESPResSo uses synchronous communication as do all other simulation programs for efficiency reasons, i.e. every node has to know without prior request which MPI communication follows. This is less robust than the asynchronous communication scheme, but the request–answer structure creates too big of an overhead, and is not needed in the integration, if everything is implemented properly.

4.4. Error handling

The Tcl command extensions of ESPResSo use the standard Tcl error return mechanism. If the error is not caught, this mechanism will terminate script execution and output a script backtrace, which, together with comprehensible error messages, allows to easily identify the source of the error.

However, as the Tcl-Interpreter exists only on the master node, ESPResSo implements its own error reporting scheme for the other nodes. Moreover, an immediate error reporting during the synchronous phase, i.e. integration, would be ineffective. Therefore, instead of terminating on an error condition, only an error message is placed in an error buffer, and the parameters are adjusted such that execution can continue. Before completing any Tcl command execution, and also during the integration loop, the master node checks regularly for errors from the slaves, and in case of an error condition, creates a regular Tcl error, which in ESPResSo is termed a background error. Background errors are reported in addition to regular Tcl errors and are reported as a string

```
background_errors 0 {<error>} {<error>}...
        1 {<error>} {<error>}...
```

Following the `background_errors` keyword, the error messages are listed, preceded by the number of the node that raised the error. If all nodes report the same error, for example, an unset timestep, then instead of repeating the error messages, a simple `<consent>` is output for each node except the master. A typical example is

```
background_errors 0 {010 time_step not set} {011 skin not set}
        {012 thermostat not initialized} 1 <consent>
```

which is the result if the integrator is started without setting up anything. Typical examples, where the messages are different from node to node, are broken bonds, which look like

```
background_errors 0 {083 bond broken between particles 1 and 0}
        1 {083 bond broken between particles 22 and 23}}
```

Each error message is assigned a unique three-digit code to support automated parsing of error messages. There are various applications of this feature, for example, testing simulation parameters for validity during tuning, or speeding up warmup up by using larger timesteps. If a bond breaks, one can simply return to an earlier timestep, simulate more carefully for some time, and then raise the timestep again. For polymer networks, this allows to reduce the warmup time considerably.

5. Documentation

ESPResSo is not intended to be a black-box-like package. Users are encouraged to try to understand its algorithms and routines, and developers are strongly advised to do so before extending it. In order to preserve the knowledge about algorithms and their physical/chemical background, it is important to provide and maintain a well structured documentation.

The hierarchical program design is reflected once more by the organization of the documentation. The first information for the user is a separated documentation of the Tcl extensions to steer the simulation, to organize the data in- and output, and to analyze the resulting data. Examples are the `part` command that modifies the particle data, or the `analyze` command that performs data analysis. It also contains documentation of Tcl procedures that serve to abbreviate common programming tasks on the script level, for example, setting up a polymer chain. In the meantime, we have also realized a through documentation of the program flow, the integration schemes, and the basic data structures to give new users an introduction to the general program design and an overview of the connection between the different modules.

The largest part of the documentation is done within the program code itself and will be automatically extracted, processed and linked to the above described parts with help of the `doxygen` program [73]. The result is a cross-referencing HTML documentation or a pdf-user manual. It is important to note that also the documentation of the Tcl commands and the underlying physical principals and algorithms is directly linked to the corresponding source code documentation.

Changes and extensions made during the development process are documented via the log function of the CVS environment (concurrent version system [74]) where the project resides in. To make important changes more visible to both users and developers, the *Release Notes* contain information about all new features, relevant changes, and important bug fixes in a condensed and easy accessible way.

6. Benchmarks and comparison

While the general aim in the development process of ESPResSo had been directed towards the flexibility and extensibility of a simulation package and its portability to a wide range of commonly used scientific computation platforms and architectures, ESPResSo is also comparable in performance with other state-of-the-art simulation codes in both single CPU and multi-processor environments. ESPResSo does not use platform-dependent optimizations, but the clever particle organization (see Section 4.1) together with algorithms like P^3M or ELC still result in a high-performance code. With Moore's Law still valid and the continuous plunging of costs for faster and more powerful computers, it is no longer economically expedient to invest human resources into developing hardware-related algorithms: Often the achievable speed-up becomes more than overcompensated by technological advancements during the time frame of the implementation, particularly since switching to different platforms or different scientific problems usually renders these optimizations useless. In this section the performance of ESPResSo will be demonstrated using several benchmarking scenarios on various hardware platforms (systems).

We begin by demonstrating the scaling of the computation time with the number of particles on a single AMD Opteron 246 processor. The benchmark scenarios are:

(b1) a simple Lennard-Jones (LJ) liquid:
The scenario consists of N particles in a cubic simulation box at a density of $\rho = 0.8442\sigma^{-3}$, with a purely repulsive Lennard-Jones potential as the only interactions between them. Simulating a (N, V, E)-ensemble, the timestep was chosen to be $\Delta t = 0.00462\tau$.
This scenario only uses the short ranged nonbonded interactions of ESPResSo. It is relatively dense, leading to a large number of simple interactions to be calculated; therefore this benchmark stresses especially the memory architecture of the used hardware platform.

(b2) a dilute electrostatic scenario:
The scenario consists of N unit charges in a cubic simulation box at a density of $\rho = 10^{-3}\sigma^{-3}$. Half of the charges are positively charged, the other half is negative. The Bjerrum length, characterizing the strength of the electrostatic interactions, is set to $\ell_B = \frac{e^2}{4\pi\epsilon_0\epsilon_s k_B T} = 2\sigma$, and again the particles also interact via a purely repulsive Lennard-Jones potential. This scenario has a temperature of $T = 1\epsilon$, which is maintained using the Langevin thermostat with a friction coefficient of $\Gamma = \tau^{-1}$, and a timestep of $\Delta t = 0.001\tau$. For the calculation of the electrostatic interactions the P^3M algorithm is used, tuned to a force error smaller than $10^{-2}\epsilon/\sigma$.
In this scenario the calculation of the long ranged electrostatic interactions dominates the computation time. Here memory bandwidth only plays a minor role compared to the floating point performance of the used CPU architecture.

(b3) Kremer–Grest polymer melt:
The scenario consists of N particles in a cubic simulation box at a density of $\rho = 0.85\sigma^{-3}$ grouped into polymer chains of 100 monomers each; in addition to a purely repulsive Lennard-Jones interaction between beads, their bonds are constructed by a FENE-spring-potential with $k_F = 30\frac{k_BT}{\sigma^2}$ and $r_F = 1.5\sigma$ (parameters taken from [40]); a timestep of $\Delta t = 0.006\tau$ with $T = 1\epsilon$ and $\Gamma = 0.5\tau^{-1}$ were chosen.
This scenario tests the ability of ESPResSo to deal with bonded interactions, and allows to compare this additional effort to the plain Lennard-Jones liquid b1 at a similarly dense density.

(b4) (N, p, T) constant pressure thermostat:
The scenario is identical to the simple Lennard-Jones scenario, but uses the (N, p, T)-integrator of ESPResSo to maintain a constant pressure of $2\epsilon/\sigma^{-2}$. The (N, p, T)-algorithm prohibits the use of Verlet lists, therefore this test shows the efficiency of the plain link cell implementation.
It also displays a typical performance difference when using ESPResSo's extensibility to modify or rewrite its core integration routine, since the (N, p, T)-algorithm is implemented as (optional) addition to the standard velocity Verlet integration scheme.

The time measurements are performed using a Tcl control script to ensure realistic numbers representing an entire integration cycle, rather than only measuring the execution times of specific parts of the simulation code. For each benchmark case, five independent pre-equilibrated configurations were integrated for thousand MD steps each, and the computation time per particle and timestep was measured. Prior to this, the cell size and, for the electrostatics test case, the P^{3}M parameters were automatically tuned using the built-in tuning functions. For all benchmarks only minimal ESPResSo components were compiled in (e.g., no electrostatics, for the neutral cases b1, b3 and b4), and compiler flags for platform-dependent optimization were used where available. Throughout this section computation times are given as wall times, as measured by the Tcl `time` command.

Fig. 4 shows the resulting computation times. As expected, the electrostatic scenario b2 is the slowest scenario, compared to the simple Lennard-Jones liquid it is slower by a factor of ≈ 3.5. The computation time for this scenario shows a typical festoon shape, which is generated by the interplay of the constant FFT computation time at fixed mesh size and an increasing real space computation time. Whenever the real space computation becomes too costly, the optimal mesh size increases, creating a larger constant offset.

For all other test cases, the computation time per particle decreases slightly for up to 1000 particles. This simply reflects that the computation time scales proportional with the number of the particles plus a small constant overhead, which becomes negligible with increasing number of particles. The data of more than 500 to 1000 particles and the necessary organizational data no longer fit into the 1 MB L2 cache of the Opteron, and more and more data has to be fetched and put back to the slower main memory. When finally the entire particle data has to be read from main memory, the computation time is around 70% larger compared to the calculation that completely uses the cache. This drop is easily understood from the fact that even linear main memory accesses are three times slower than cache accesses, see Section 4.1. Nevertheless, 1.4 μs per particle and timestep is still a very good performance and comparable to other simulation codes. The cache effect is not visible in the electrostatics test case, since the slowdown through the grid discretization is much stronger.

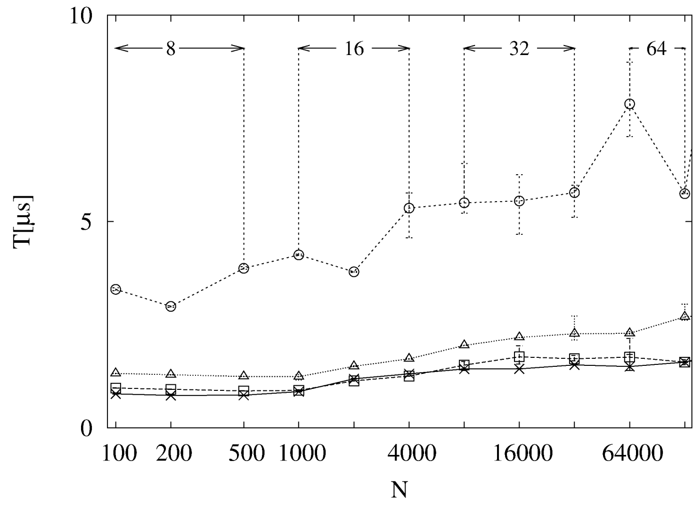

Fig. 4. Computation time T in microseconds per particle and timestep of ESPResSo for various benchmark test cases with different numbers of particles N, each run on a single AMD Opteron 246 processor. The symbols stand for the test cases b1 (crosses), b2 (circles), b3 (squares) and b4 (triangles). The arrows denote the P^{3}M mesh size used, e.g., 8 for up to 500 particles, 16 for up to 4000 particles and so on. For details on the test cases, see text.

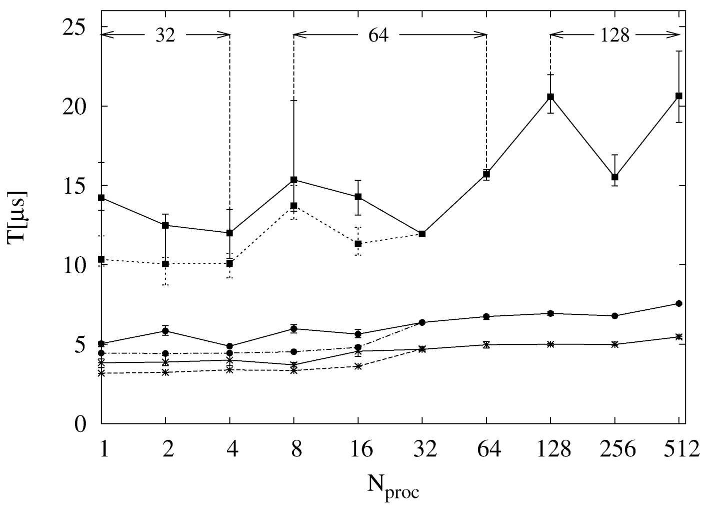

Fig. 5. Computation time T in microseconds per particle and timestep of ESPResSo for the benchmark test cases b1 (stars), b2 (squares), b4 (circles) on an IBM Regatta system of up to 16 pSeries 690 with different numbers of processors N_{proc} and 4000 particles per processor. For each test case, the dotted, lower lines denote results for an otherwise unloaded 32 processor machine, while the solid, upper lines represent data for a fully loaded machine.

It is remarkable that the additional computation of the bonded interactions in the Kremer–Grest test case does not lead to a significant increase in the computation time. This shows that, once the organizational part of the integration is done and the particle data is loaded into the fast L1 cache of the CPU, the computation time spent in the actual force calculation is almost negligible.

The approximately constant offset between b4 and the two fastest test cases (b1 and b3) originates in the (N, p, T)-algorithm itself, which requires an additional amount of operations for each particle (i.e. pressure evaluations, coordinate rescaling, and thermostat application) without further accesses to stored data; consequently, the computation time increases compared to b1 and b3 while displaying the same cache-related behavior on the amount of particles N in the simulation.

For a parallel simulation code the scaling with the number of processors is as important as the scaling with the number of particles. In Fig. 5 we demonstrate the scaling of ESPResSo for the simple Lennard-Jones liquid test case b1, the electrostatics test case b2, and the (N, p, T) test case b4 on an IBM Regatta system consisting of up to 16 pSeries 690 system with 32 Power4 CPUs running at 1.3 GHz each. The Kremer–Grest test case b3 was not repeated on the multiprocessor system, as it is very similar to the Lennard-Jones test case (see Fig. 4). The number of particles was chosen proportional to the number of processors, namely 4000 particles per processor. For less than 32 processors, a single machine was used for two runs, one on an otherwise unloaded machine and one on a fully loaded machine (which is the more realistic scenario, but naturally generates larger error bars and may increase the computation time if memory access is the bottleneck). For more than 32 processors, we always used entire 32 processor machines, interconnected by an IBM federation switch.

From the computation times given in Fig. 5, it is easy to calculate the speedup factors for ESPResSo. The speedup factor for N_p processors is defined as the computation time for a single CPU divided by the computation time for N_p processors. For testsystem b1, the factor is between 95 and 105% for 1 to 16 processors, 85% for 32 processors, and slowly drops to 70% for 512 processors.

Both b1 and b4 scale extremely well with the number of processors. For the b2 scenario ESPResSo shows the typical festoon behavior, which was already observed in the single CPU benchmarks, and which is due to the complex interplay of FFT time and real space calculation time. It should be noted that ESPResSo scales extremely well up to 512 processors or 2 million particles, even for the electrostatic scenario, and there is no fundamental reason that ESPResSo should not able to run on even larger supercomputers.

In the b2 test case, the Regatta system switches faster to higher grid sizes, which simply reflects that the Regatta FFTW implementation performs extremely well. Again, the additional computational effort required for the (N, p, T) is responsible for a constant (i.e. N-dependent) decrease of that scenario's performance compared to the LJ-liquid's case. For the b1 and b4 scenario, there is a noticeable computation time difference between a fully loaded and an unloaded machine. This is a result of the shared memory architecture of the Regatta system: The CPUs are organized in dual core units with a small shared L2 cache of only 1.44 MB. Four of these dual core units form a processor board with a total of eight processors, which share an additional L3 cache of 32 MB. If all 32 processors are loaded, the L2 cache per CPU is 768 kB, which is not enough for 4000 particles, and the CPU has to resort to the L3 cache. However, if only one processor per dual core is active (as is the case for $N_{\text{proc}} \leqslant 16$ on an otherwise unloaded machine), then it can use the full 1.44 MB L2 cache, which is sufficient for the data of 4000 particles. However, in a production environment normally all 32 CPUs will be loaded, so that the numbers for the loaded system are more realistic.

In contrast to the memory organization of the Regatta system, the federation switch shows a high performance, and does not seem to add a noticeable additional performance loss for less than 16 machines. Even for 16 machines, the impact is only small. Therefore, it should be possible to simulate scenarios of several million charged particles using ESPResSo on a sufficiently large IBM Regatta system. This is mainly due to the small latency of the federation switch, since the domain decomposition cell system

Table 1
Computation time T in microseconds per particle and timestep for the b1 and b2 test cases with 4000 (top) and 16000 (bottom) particles on the platforms described in the text

System	T_{b1} [μs]	N_{cells}	T_{b2} [μs]	N_{cells}	Mesh
s1	1.31 (1.31, 1.31)	10	5.32 (4.60, 5.69)	8	16
s2	1.58 (1.57, 1.59)	10	4.87 (4.85, 4.89)	8	16
s3	3.35 (3.34, 3.35)	10	11.36 (10.50, 12.72)	7.6 (6, 8)	16
s4	2.18 (1.99, 2.48)	10.4 (10, 11)	7.55 (5.66, 10.24)	7.4 (7, 8)	16
s5	2.27 (2.26, 2.27)	10	8.69 (7.39, 10.64)	7.6 (7, 8)	16
s6	3.83 (3.54, 4.08)	10.7 (10, 11)	14.23 (13.43, 16.45)	11.8 (7, 14)	28.0 (16, 32)
s7	2.37 (2.37, 2.38)	11.2 (11, 12)	7.58 (7.28, 8.66)	13.6 (11, 15)	32
s8	2.26 (2.23, 2.29)	10.8 (10, 11)	9.75 (9.70, 9.82)	8	16
s1	1.53 (1.41, 1.83)	15.9 (15, 16)	4.77 (4.72, 4.91)	14	32
s2	1.63 (1.63, 1.64)	16	5.79 (5.76, 5.81)	14	32
s3	4.11 (3.47, 5.44)	15.6 (15, 16)	15.88 (15.21, 16.51)	13.6 (12, 14)	32
s4	3.14 (3.06, 3.17)	15	10.19 (7.58, 12.05)	13.4 (13, 14)	32
s5	3.05 (2.82, 3.36)	15.8 (15, 17)	9.27 (8.48, 11.32)	13 (10, 14)	32
s6	5.52 (4.26, 6.64)	18.3 (15, 17)	18.84 (14.06, 22.47)	11.6 (11, 13)	32
s7	2.60 (2.56, 2.63)	17.8 (17, 18)	7.16 (7.07, 7.39)	14	32
s8	3.01 (2.93, 3.07)	16.8 (16, 17)	11.48 (11.34, 11.60)	14	32

N_{cells} denotes the used cell grid sizes (due to the symmetry of the test cases, the cell grid is always cubic, i.e. $N_{\text{cells}} \times N_{\text{cells}} \times N_{\text{cells}}$). For the b2 test case, the Coulomb mesh size is shown as well. The numbers are given as average(min, max) of the five runs performed for each system. If minimum and maximum are not given, all five measurements produced the same value.

of ESPResSo is primarily communication intensive. For N_p processors, $12N_p$ communications per time step are necessary to update the ghost information, of which $N_p/2$ can be done in parallel. Bandwidth in turn is not as important, since the amount of data transferred is quite small; only particles in the ghost shells at the processor boundaries are affected, which are normally less than 10% of all particles.

Now that we have shown the scaling of ESPResSo for some of the built-in algorithms on a single CPU and a multiprocessor system, we finally want to present results from different architectures, namely

(s1) dual AMD Opteron 246, 2 GB RAM, S.u.S.E.-Linux 9.1, gcc 3.3 [75],
(s2) single AMD Athlon64 3200+, 1 GB RAM, S.u.S.E.-Linux 9.1, gcc 3.3,
(s3) dual AMD AthlonMP 2000+, 1 GB RAM, S.u.S.E.-Linux 9.0, gcc 3.1,
(s4) dual Intel Xeon 2.8 GHz, 3 GB RAM, RedHat 9.0, icc v8.0 [76],
(s5) dual Apple G5 2.0 GHz, 1 GB RAM, MacOS X (Darwin 7.4.0), gcc 3.3,
(s6) IBM pSeries 690 Turbo, 32-way Power4 1.3GHz, 64 GB RAM, AIX 5.2, IBM xlc V6,
(s7) IBM pSeries 655, 8-way Power4 1.7 GHz, 16 GB RAM, AIX 5.2, IBM xlc v6,
(s8) dual Alpha 21264 833 MHz, 3 GB RAM, Tru64 V5.1A, Compaq C V6.4.

Again, the benchmarking cases b1 and b2 were run five times with 4000 and 16 000 particles on each architecture, measuring the computation time per particle and timestep, as well as the used number of cells N_{cells} and mesh points per dimension. The majority of the systems are multi processor systems with varying numbers of processors, however for comparability all tests were done on only *one* of the CPUs. All other CPUs were loaded with other simulations in a typical production environment. Although this perturbs the measured computation times a little bit (for systems with shared access to the memory banks, e.g., s3 or s6, but *not* for systems such as s1 which have a n-way connection between CPU and RAM), it gives the best picture of what one can expect when running ESPResSo for production. The results are shown in Table 1, giving an overview of the efficiency of ESPResSo on the various platforms.

For scenario b1 and 4000 particles, the maximal possible number of cells is 14, for scenario b2 8 cells with a FFT mesh size of 16 and a charge assignment order of 3, and 15 cells for a mesh size of 32 and order 3. The optimal grid sizes for the test cases are slightly smaller, and correspond to skin sizes of 0.3–0.5σ. For 16 000 particles, the maximal number of cells is 23, and the chosen cell sizes correspond to skin sizes of 0.3–0.6σ. Despite the fact that the number of cells is chosen automatically by a simple golden section search, the resulting cell sizes correspond to skins which are only slightly larger than the values typically used by other simulation codes, which range from 0.2σ to 0.4σ. The reason for this is, that the link cell implementation of ESPResSo requires more memory operations during the particle reorganization, making it slower compared to other implementations, while the building of the Verlet list and the force calculation are somewhat faster. The larger skin sizes that are optimal for ESPResSo reflect this, since a larger skin means less frequent particle reorganization steps.

Systems s1 through s5 favor smaller cell grids compared to systems s6 through s8, especially for 16 000 particles. Since the cell grid size represents a trade off between an increased number of particle moves for large numbers of cells, and an increased number of particle distance calculations for smaller grids, this means that memory operations are fast compared to floating point operations on the latter systems. This is an effect of the different cache sizes of the used systems: Systems s1 through s5 have L2 cache sizes of at most 1 MB, and only system s4 has an additional L3 cache of 2 MB per CPU. Therefore only for system s4 4000 particles fit into the L3 cache. In contrast, systems s6 and s7 have 1.44 MB L2 cache per two CPUs and 32 MB L3 cache per eight CPUs, and system s8 even has an 8 MB L2 cache. Therefore the latter systems easily fit the data of 16 000 particles into their caches, increasing the memory performance dramatically.

The outstanding performance of the AMD 64-bit systems s1 and s2 is due to their large memory bandwidth and the additional 64-bit registers, which can host floating point numbers and therefore speed up the force calculation. In contrast to this, the performance of the IBM pSeries 690 system is rather weak. The much better performance of the pSeries 655, which has a faster memory bus, as well as the multiprocessor results for the unloaded machines suggest that the primary reason for this lies in the memory performance of the pSeries architecture, despite the large caches. For the electrostatic test case b2, the efficiency of the FFTW routine is important, which on the Regatta is much better than the performance of the other ESPResSo components.

Compared to high performance MD simulation codes like GROMACS or LAMMPS, ESPResSo excels through its parallelization where it is much faster than, e.g., GROMACS, although under certain conditions it may also be somewhat slower, depending on the used hardware and the benchmark scenario employed. The main reason for such a performance drop in those cases is the flexibility of the cell systems. As described before, different cell systems require different force evaluation strategies. Therefore the force calculation routines for the potentials are used in several different loops, and therefore need a simple call interface. This interface creates some overhead, mainly due to the inability of current compilers to keep the force array values in registers.

Since not all particles are stored in a single list in ESPResSo, the only way to allow easy access to the particle data, is the organization of the particle data in a single structure, as described before. However, an organization of the different properties into different arrays, such as in GROMACS or LAMMPS, would be computationally more efficient. The reason is that current processors read and write data into their L2 cache always in 32 byte cache lines, but the position, velocity or force data consists of three doubles or 24 bytes. Therefore a loop adding the force to the velocity will have to read 64 bytes into the L2 cache, but only 48 of them will be actually used, and similarly, 32 bytes have to be written, also only 24 of them actually changed. If the force and velocity data is organized consecutively, this is no problem, since the additionally read data belongs to the next particle. But in the case of a single particle structure, this additionally read data are other informations on the same particle, which for the current loop are not used. Since memory accesses are costly, this is responsible for a considerable performance drop. But in the C language, it is impossible to write a simple interface for the force evaluation, if it has to get its data from various dynamically allocated arrays. We decided to favor simplicity and flexibility over performance.

However, looking at the aforementioned unavoidable hardware restrictions, the impact of compiler optimization and the continuing improvements expected therein, and the competitive performance of ESPResSo even for modified and customized scenarios, our design goals from Section 2 seem once again confirmed.

7. Conclusion and outlook

Several publications have already appeared that used ESPResSo for data production [13,14,34,49,77–82]. As of this writing the ESPResSo-package continues to undergo significant enlargement. We are currently implementing a dipolar Ewald sum [83] and will also add a dipolar P^3M version, which will enhance the capabilities to simulate ferrofluids or dipolar fluids like simple water models.

For the dynamics of soft matter systems it is often necessary to include hydrodynamic interactions. Since in practice one cannot consider all molecular details of the systems, this can be achieved on a coarse grained level by coupling the solvent degrees of freedom to the simulated particles. We have almost finished the implementation of an advanced lattice Boltzmann algorithm [84] that has already proven its usefulness in polymer dynamics simulations [85] and electrophoresis simulations of charged colloids [86,87].

Clever Monte Carlo strategies are lacking so far. Although easily implementable on the Tcl-level [13], where, for example, a hybrid Monte Carlo technique has been successfully employed [88,89], other, more efficient sampling methods, are planned for the future.

Other features like the `rattle` algorithm for constraint dynamics, and more potentials suited for atomistic force fields are already in progress and will continue to enable ESPResSo to handle atomistic simulations. The ultimate goal is to implement multiscale simulation strategies which will allow probing different length and time scales within one single simulation run of ESPResSo.

There are many more improvements planned for the years to come, and hopefully the capabilities of ESPResSo will grow even further once more researchers take up our idea and contribute to the package by using, customizing, and extending it. Our hope is to attract further developers that can contribute to our project, while keeping the anticipated program structure simple, extensible, well documented, and easy to use. We realize that this is a challenging goal to achieve, although the past two years, in which ESPResSo left its alpha stage and first external collaborations have started, encouraged us that this challenge can be mastered.

Acknowledgements

As we have mentioned, ESPResSo has started as and still is a team project, and it would not have been possible to create it without all the help and enthusiasm of many people.

We thank D. Antypov, V. Ballenegger, I. Cooke, B. Dünweg, D. Galperin, V. Lobaskin, F. Mühlbacher, I. Pasichnyk, M. Praprotnik, M. Sayar, T. Schürg, T. Stühn, S. K. Sukumaran and Z. Wang for their various contributions, including many hours of discussions and thorough testing of the code. ESPResSo's webpage was re-written and greatly enhanced by N. Binz who also added a lot of helpful scripts, all of which we are very grateful for.

For preceding work and quite a number of clever ideas about efficient programming we would like to thank M. Pütz and M. Deserno.

We also want to thank for the contributions which are already made by people from other groups, namely A. Heuer and A. Maitra from the University of Münster, and O. Lenz, U. Schiller and F. Schmid from the University of Bielefeld.

H.L. would like to thank M. Doi and his group for their effort to implement the OCTA-ESPResSo interface and their hospitality.

Financial contributions via grants HO-1108/8-3, HO-1108/11-1, BMBF 03N 6500, and the DFG within the SFB 625 and TR 6 are gratefully acknowledged.

Finally we would like to thank K. Kremer and his theory group at the Max-Planck-Institute for Polymer Research in Mainz for their continuing support of this project over all the years.

Appendix A. Sample script

To demonstrate the use of the Tcl script language, here an ESPResSo control script for the (N, V, T) simulation of a Lennard-Jones liquid is given. With a small additional header ("she-bang") the script can also be called directly from the command line like any other program, which is very convenient for submitting several jobs at once. First some Tcl variables are set which will be used later in the program to determine the simulation parameters:

```
# size of the cubic simulation box
set box_length 10.7437
# density of the liquid
set density 0.7
```

The next lines define some parameters of the simulation, the timestep, box length and the skin depth. The latter parameter is needed for the link-cell algorithm (see Section 3.3), but has no influence on the physics of the simulated liquid.

```
setmd time_step 0.01
setmd skin      0.4
setmd box_l $box_length $box_length $box_length
```

Now particles are put into the simulation box. The first two lines show how mathematical expressions can be used inside Tcl. Here the number of particles is determined from the box size and the density. The particles are added particle for particle at random positions in a loop, using the built-in random number generator of ESPResSo. Here only the properties "position" and "type" are set, other properties are, e.g., velocity or charge.

```
set volume [expr $box_l*$box_l*$box_l]
set n_part [expr floor($volume*$density)]

for {set i 0} { $i < $n_part } {incr i} {
    set posx [expr $box_l*[t_random]]
    set posy [expr $box_l*[t_random]]
    set posz [expr $box_l*[t_random]]

    part $i pos $posx $posy $posz type 0
}
```

The Langevin thermostat (see Section 3.2) is activated for a temperature of $1kT$ and a friction constant $\Gamma = 1/\tau$ by the command

```
set temp 1
set gamma 1
thermostat langevin $temp $gamma
```

A purely repulsive Lennard-Jones interaction between all particles of type 0, which are all in our case, is defined by

```
set lj1_eps   1.0
set lj1_sig   1.0
set lj1_cut   1.12246
set lj1_shift [expr 0.25*$lj1_eps]
inter 0 0 lennard-jones $lj1_eps $lj1_sig $lj1_cut $lj1_shift 0
```

This interaction is used widely in computer simulations and is a smooth approximation of a hard core interaction between two spheres of diameter 1.

Now a simulation loop for 1 million timesteps could look like

```
for {set i 0} { $i < 1000 } { incr i} {
    puts ''step $i ftime=[setmd time] energy=[analyze energy total]''
    puts ''temp = [expr [analyze energy kinetic]/(1.5*[setmd n_part])]''
    integrate 1000
}
```

Every thousand timesteps the simulation time, total energy and the current temperature are printed out. Of course this is not enough for a real simulation, as one normally wants to write out simulation data to configuration files. To this aim ESPResSo has commands to write simulation data to a Tcl stream. The format written by these commands is called *blockfile* in the following. One could add the following lines into the simulation to write configuration files “config_0” through “config_999”:

```
set f [open ''config_$i'' ''w'']
blockfile $f write variable {time_step skin}
blockfile $f write tclvariable {box_length density}
set temp [expr [analyze energy kinetic]/(1.5*[setmd n_part])]
puts $f ''\{energy [analyze energy total] $temp\}''
blockfile $f write particles {id pos type}
close $f
```

The created files “config_...” look like

```
{variable
        {time_step 0.01}
        {skin 0.4}
}
{tclvariable
        {box_length {10.7437}}
        {density {0.7}}
}
{energy 5380.6 1.00005}
{particles {id pos type}
        {0 11.5573 8.87179 4.80079 0}
        {1 7.04209 2.96786 3.74511 0}
        {2 8.08042 4.91621 6.53135 0}
        .
        .
        .
        {864 3.64869 8.02398 3.13255 0}
        {865 7.66632 14.4887 1.31884 0}
        {866 -0.654808 11.9669 0.43845 0}
        {867 6.43409 5.75895 5.46044 0}
}
```

As one can see, the format of a blockfile corresponds to Tcl lists of tagged data sets, which are called *blocks*, and allows for adding own data sets like the “energy” block in this case. Depending on the simulation, much more complex observables can be written out here for an easy inspection during the simulation run.

Reading in the blockfiles is automated in ESPResSo. The tagged structure of a blockfile allows an easy identification of the portions ESPResSo knows about and proper parsing. The command

```
set f [open ''config_999'' ''w'']
while { [blockfile $f read auto] != ''eof'' } {}
close $f
```

will read in the variables time_step and skin and the Tcl variables box_length and density back in, as well as all particles, while the energy entry is unknown and ignored.

The simulation as presented above will very likely not run smoothly, since the particles are placed randomly and therefore could overlap. The Lennard-Jones potential is singular for vanishing particle distance, therefore overlapping particles have an extremely high interaction energy. The high initial potential energy will in turn accelerate the particles to velocities larger than what can be treated with the chosen timestep, and the simulation might crash. To avoid this, ESPResSo allows to cap the Lennard-Jones interaction, i.e. below a certain distance the potential only grows linearly while the Lennard-Jones force is constant. This force cap is gradually raised in an equilibration loop which might look like

```
set cap 10
while {[analyze mindist] < 0.95} {
    inter ljforcecap $cap
    integrate 1000
    incr cap 30
    inter ljforcecap 0
```

This loop will gradually increase the force cap until the minimal distance is larger than 0.95 and then switches off the force cap.

The code example given before is obviously a very simple one. More complicated simulation scripts can extend over thousands of lines of code and contain automated adaption of parameters or online analysis. Parameters can be changed arbitrarily during the simulation process, as needed for, e.g., simulated annealing. Another example is a simulation of a polymer which is fixed to a surface with one end, while one pulls at the other end. The possibility to perform non-standard simulations without the need of modifications to the simulation core was one of the main reasons why we decided to use a script language for controlling the simulation core.

References

[1] R.A.L. Jones, Soft Condensed Matter, Oxford University Press, Oxford, 2002.
[2] L. Treloar, The Physics of Rubber Elasticity, Clarendon Press, Oxford, 1986.
[3] C. Holm, P. Kékicheff, R. Podgornik (Eds.), Electrostatic Effects in Soft Matter and Biophysics, NATO Science Series II—Mathematics, Physics and Chemistry, vol. 46, Kluwer Academic Publishers, Dordrecht, 2001.
[4] M.J. Stevens, K. Kremer, The nature of flexible linear polyelectrolytes in salt free solution: A molecular dynamics study, J. Chem. Phys. 103 (4) (1995) 1669–1690.
[5] C. Holm, Efficient methods for long range interactions in periodic geometries plus one application, in: N. Attig, K. Binder, H. Grubmüller, K. Kremer (Eds.), Computational Soft Matter: From Synthetic Polymers to Proteins, in: NIC Series, vol. 23, Research Centre Jülich, 2004.
[6] A. Arnold, C. Holm, Efficient methods to compute long range interactions for soft matter systems, in: C. Holm, K. Kremer (Eds.), Advanced Computer Simulation Approaches for Soft Matter Sciences, in: Advances in Polymer Sciences, vol. II, Springer, 2005, pp. 59–109, 185.
[7] J. Norberg, L. Nilsson, Advances in biomolecular simulations: methodology and recent applications, Quart. Rev. Biophys. 36 (3) (2003) 257–306.
[8] M. Deserno, C. Holm, How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines, J. Chem. Phys. 109 (1998) 7678.
[9] M. Deserno, C. Holm, How to mesh up Ewald sums. II. An accurate error estimate for the p3m algorithm, J. Chem. Phys. 109 (1998) 7694.
[10] A. Arnold, C. Holm, MMM2D: A fast and accurate summation method for electrostatic interactions in 2D slab geometries, Comput. Phys. Comm. 148 (3) (2002) 327–348.
[11] A. Arnold, J. de Joannis, C. Holm, Electrostatics in periodic slab geometries. I, J. Chem. Phys. 117 (2002) 2496–2502.
[12] J. de Joannis, A. Arnold, C. Holm, Electrostatics in periodic slab geometries. II, J. Chem. Phys. 117 (2002) 2503–2512.
[13] B.A. Mann, R. Everaers, C. Holm, K. Kremer, Scaling in polyelectrolyte networks, Europhys. Lett. 17 (2004) 786–793.
[14] B.A. Mann, C. Holm, K. Kremer, Swelling behaviour of polyelectrolyte networks, J. Chem. Phys. 122 (2005) 154903.
[15] W. Tschöp, K. Kremer, J. Batoulis, T. Bürger, O. Hahn, Simulation of polymer melts. I. Coarse-graining procedure for polycarbonates, Acta Polymer. 49 (1998) 61–74.
[16] W. Tschöp, K. Kremer, J. Batoulis, T. Bürger, O. Hahn, Simulation of polymer melts. II. From coarse-grained models back to atomistic description, Acta Polymer. 49 (1998) 75–79.
[17] K. Kremer, F. Müller-Plathe, Multiscale simulation in polymer science, Mol. Sim. 28 (2002) 729–750.
[18] N. Boghossian, O. Kohlbacher, H.-P. Lenhof, BALL: Biochemical ALgorithm Library, Research report, Max-Planck-Institut für Informatik, Saarbrücken, 1999.
[19] C.J. Lejdfors, GISMOS: Graphics and Interactive Steering of MOlecular Simulations, Homepage, 1998, URL: http://www.teokem.lu.se/gismos/.
[20] W.F. van Gunsteren, GROMOS96: GROningen MOlecular Simulation computer program package, Homepage, 1996, URL: http://www.igc.ethz.ch/gromos/.
[21] E. Lindahl, B. Hess, D. van der Spoel, GROMACS 3.0: A package for molecular simulation and trajectory analysis, J. Mol. Mod. 7 (2001) 306–317.
[22] S.J. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.
[23] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kale, R. Skeel, K. Schulten, NAMD—a parallel, object-oriented molecular dynamics program, Internat. J. Supercomput. Ap. 10 (4) (1996) 251–268.

[24] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, K. Schulten, NAMD2: Greater scalability for parallel molecular dynamics, J. Comput. Phys. 151 (1999) 283–312.
[25] M. Pütz, A. Kolb, Optimization techniques for parallel molecular dynamics using domain decomposition, Comput. Phys. Comm. 113 (1998) 145–167.
[26] D. Pearlman, D. Case, J. Caldwell, W. Ross, I.T.E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, AMBER: Assisted Model Building with Energy Refinement—A computer program for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to elucidate the structures and energies of molecules, Comput. Phys. Comm. 91 (1995) 1–41.
[27] T. Straatsma, E. Aprà, T. Windus, E. Bylaska, W. de Jong, S. Hirata, M. Valiev, M.T. Hackler, L. Pollack, R.J. Harrison, M. Dupuis, D. Smith, J. Nieplocha, T.V.M. Krishnan, A.A. Auer, E. Brown, G. Cisneros, G.I. Fann, H. Fruchtl, J. Garza, K. Hirao, R. Kendall, J. Nichols, K. Tsemekhman, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffe, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong, Z. Zhang, NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.6, Pacific Northwest National Laboratory, Richland, WA 99352-0999, USA, 2004.
[28] W. Smith, T. Forester, Dl_poly_2.0: A general-purpose parallel molecular dynamics simulation package, J. Molec. Graphics 14 (1996) 136.
[29] M. Doi, OCTA: Growth for the Future, Homepage, 2003, URL: http://octa.jp/OCTA/whatsOCTA.html.
[30] A. Arnold, B.A. Mann, H.-J. Limbach, C. Holm, ESPResSo—An Extensible Simulation Package for Research on Soft Matter Systems, in: K. Kremer, V. Macho (Eds.), Forschung und wissenschaftliches, Rechnen 2003, in: GWDG-Bericht, vol. 63, Gesellschaft für wissenschaftliche Datenverarbeitung mbh, Göttingen, Germany, 2004, pp. 43–59.
[31] INFM Democritos, Quantum-Espresso, Homepage, 2005, URL: http://www.democritos.it/scientific.php.
[32] GPL, GNU General Public License, Homepage, 2004, URL: http://www.gnu.org/copyleft/gpl.html.
[33] J. Rottler, A. Maggs, Local molecular dynamics with Coulombic interactions, Phys. Rev. Lett. 93 (17) (2004) 170201.
[34] I. Pasichnyk, B. Dünweg, Coulomb interactions via local dynamics: A molecular-dynamics algorithm, J. Phys. Condens. Matter 16 (38) (2004) 3999–4020.
[35] Tcl/Tk, Tool Command Language/ToolKit, Homepage, 2003, URL: http://tcl.activestate.com/.
[36] A. Bunker, B. Dünweg, Parallel excluded volume tempering for polymer melts, Phys. Rev. E 63 (2).
[37] FFTW, Fastest Fourier Transform in the West, Homepage, 2003, URL: http://www.fftw.org/.
[38] LAM/MPI, Local Area Multicomputer Message Passing Interface, Homepage, 2004, URL: http://www.lam-mpi.org/.
[39] MPICH, Message Passing Interface CHameleon, Homepage, 2004, URL: http://www-unix.mcs.anl.gov/mpi/mpich/.
[40] K. Kremer, G. Grest, Dynamics of entangled linear polymer melts: A molecular-dynamics simulation, J. Chem. Phys. 92 (1990) 5057.
[41] D.C. Rapaport, The Art of Molecular Dynamics Simulation, second ed., Cambridge University Press, 2004.
[42] D. Frenkel, B. Smit, Understanding Molecular Simulation, second ed., Academic Press, San Diego, 2002.
[43] N. Martys, R. Mountain, Velocity Verlet algorithm for dissipative-particle-dynamics-based models of suspensions, Phys. Rev. E 59 (3) (1999) 3733–3736.
[44] F. Müller-Plathe, Reversing the perturbation in non-equilibrium molecular dynamics: An easy way to calculate the shear viscosity of fluids, Phys. Rev. E 59 (1999) 4894–4899.
[45] T. Soddemann, B. Dünweg, K. Kremer, Dissipative particle dynamics: A useful thermostat for equilibrium and nonequilibrium molecular dynamics simulations, Phys. Rev. E 68 (2003) 046702.
[46] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, J.R. Haak, Molecular dynamics with coupling to a heat bath, J. Chem. Phys. 81 (1984) 3684–3690.
[47] P. Español, P. Warren, Statistical mechanics of dissipative particle dynamics, Europhys. Lett. 30 (1995) 191.
[48] A. Kolb, B. Dünweg, Optimized constant pressure stochastic dynamics, J. Chem. Phys. 111 (10) (1999) 4453–4459.
[49] I.R. Cooke, K. Kremer, M. Deserno, Efficient tunable generic model for fluid bilayer membranes, Phys. Rev. E 72 (2004) 011506.
[50] J.D. Weeks, D. Chandler, H.C. Andersen, Role of repulsive forces in determining the equilibrium structure of simple liquids, J. Chem. Phys. 54 (1971) 5237.
[51] T. Soddemann, B. Dünweg, K. Kremer, A generic computer model for amphiphilic systems, European J. Phys. E 6 (2001) 409.
[52] A.D. Buckingham, P.W. Fowler, A model for the geometries of van der Waals complexes, Canad. J. Chem. 63 (1985) 2018–2025.
[53] J.G. Gay, B.J. Berne, Modification of the overlap potential to mimic a linear site-site potential, J. Chem. Phys. 74 (6) (1981) 3316–3319.
[54] P. Debye, E. Hückel, Zur Theorie der Elektrolyte. I. Gefrierpunktserniedrigung und verwandte Erscheinungen, Phys. Z. 24 (9) (1923) 185.
[55] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, IOP, London, 1988.
[56] T. Darden, D. York, L. Pedersen, Particle mesh Ewald: An $n\log(n)$ method for Ewald sums in large systems, J. Chem. Phys. 98 (1993) 10089.
[57] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L. Pedersen, A smooth particle mesh Ewald method, J. Chem. Phys. 103 (1995) 8577.
[58] A. Arnold, C. Holm, A novel method for calculating electrostatic interactions in 2d periodic slab geometries, Chem. Phys. Lett. 354 (2002) 324–330.
[59] A. Arnold, C. Holm, MMM1D: A method for calculating electrostatic interactions in 1D periodic geometries, J. Chem. Phys. 123 (2005) 144103.
[60] A. Arnold, Berechnung der elektrostatischen Wechselwirkung in $2\mathrm{d}+\mathrm{h}$ periodischen Systemen, Diploma thesis, Johannes Gutenberg-Universität, May 2001; URL: http://www.mpip-mainz.mpg.de/www/theory/phd_work/dipl-arnold.ps.gz.
[61] A. Arnold, Computer simulations of charged systems in partially periodic geometries, Ph.D. thesis, Johannes Gutenberg-University, Mainz, Germany, August 2004.
[62] J. Lekner, Summation of Coulomb fields in computer simulated disordered systems, Physica A 176 (1991) 485–498.
[63] E.R. Smith, Electrostatic energy in ionic crystals, Proc. Roy. Soc. London A 375 (1981) 475–505.
[64] A. Maggs, V. Rosseto, Local simulation algorithms for Coulombic interactions, Phys. Rev. Lett. 88 (2002) 196402.
[65] VMD, Visual Molecular Dynamics, Homepage, 2003, URL: http://www.ks.uiuc.edu/Research/vmd/.
[66] F. Oded, "Water free" computer model for lipid bilayer membranes, J. Chem. Phys. 119 (1) (2003) 596–605.
[67] U. Wolff, Monte Carlo errors with less errors, Comput. Phys. Comm. 156 (2004) 143–153.
[68] R.W. Hockney, J.W. Eastwood, Computer Simulations using Particles, McGraw-Hill, New York, 1981.
[69] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, first ed., Oxford Science Publications, Clarendon Press, Oxford, 1987.
[70] G.S. Grest, B. Dünweg, K. Kremer, Vectorized link cell Fortran code for molecular-dynamics simulations for a large number of particles, Comput. Phys. Comm. 55 (3) (1989) 269–285.
[71] M. Rubinstein, S. Panyukov, Nonaffine deformation and elasticity of polymer networks, Macromolecules 30 (25) (1997) 8036–8044.
[72] R. Everaers, Constrained fluctuation theories of rubber elasticity: General results and an exactly solvable model, Eur. Phys. J. B 4 (3) (1998) 341–350.
[73] Doxygen, Doxygen—A documentation generation system, 2005, URL: http://www.doxygen.org/.
[74] CVS, Concurrent Versions System, Homepage, 2003, URL: http://www.cvshome.org/.
[75] gcc, Gnu compiler suite, 2002–2004, URL: http://gcc.gnu.org/.

[76] icc, Intel c/c++ compiler for Linux, 2003–2004, URL: http://www.intel.com/software/products/compilers/clin.
[77] H.J. Limbach, M. Sayar, C. Holm, Polyelectrolyte bundles, J. Phys. Condens. Matter 16 (22) (2004) S2135–S2144.
[78] H.J. Limbach, C. Holm, K. Kremer, Computer simulations of the “hairy rod” model, Macromolecular Chem. Phys. 206 (2005) 77–82.
[79] A. Naji, A. Arnold, C. Holm, R.R. Netz, Attraction and unbinding of like-charged rods, Europhys. Lett. 67 (2004) 130–136.
[80] R. Everaers, S.K. Sukumaran, G.S. Grest, C. Svaneborg, A. Sivasubramanian, K. Kremer, Rheology and microscopic topology of entangled polymeric liquids, Science 303 (5659) (2004) 823–826.
[81] S.K. Sukumaran, G.S. Grest, K. Kremer, R. Everaers, J. Polym. Sci. B: Pol. Phys. 43 (2005) 917–933.
[82] I. Kulic, H. Mohrbach, V. Lobaskin, R. Thaokar, H. Schiessel, Apparent persistence length renormalization of bent DNA, Phys. Rev. E 72 (2005) 041905.
[83] Z.W. Wang, C. Holm, Estimate of the cutoff errors in the Ewald summation for dipolar systems, J. Chem. Phys. 115 (2001) 6277–6798.
[84] P. Ahlrichs, B. Dünweg, Lattice-Boltzmann simulation of polymer-solvent systems, Int. J. Modern Phys. C 9 (8) (1998) 1429–1438.
[85] P. Ahlrichs, B. Dünweg, Simulation of a single polymer chain in solution by combining lattice Boltzmann and molecular dynamics, J. Chem. Phys. 111 (17) (1999) 8225–8239.
[86] V. Lobaskin, B. Dünweg, A new model for simulating colloidal dynamics, New J. Phys. 6 (2004) 54.
[87] V. Lobaskin, B. Dünweg, C. Holm, Electrophoretic mobility of a charged colloidal particle: A computer simulation study, J. Phys. Condens. Matter 16 (38) (2004) S4063–S4073.
[88] S. Duane, A.D. Kennedy, B.J. Pendleton, D. Roweth, Hybrid Monte-Carlo, Phys. Lett. B 195 (2) (1987) 216–222.
[89] M. Sayar, C. Holm, Unpublished.