Comparison of scalable fast methods for long-range interactions

Axel Arnold, Florian Fahrenberger, Christian Holm, and Olaf Lenz
Institute for Computational Physics, University of Stuttgart, Stuttgart, Germany

Matthias Bolten
Department of Mathematics and Science, University of Wuppertal, Wuppertal, Germany

Holger Dachsel, Rene Halver, and Ivo Kabadshow
Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany

Franz Gähler
Faculty of Mathematics, Bielefeld University, Bielefeld, Germany

Frederik Heber and Julian Iseringhausen
Institute for Numerical Simulation, University of Bonn, Bonn, Germany

Michael Hofmann
Department of Computer Science, Chemnitz University of Technology, Chemnitz, Germany

Michael Pippig and Daniel Potts
Department of Mathematics, Chemnitz University of Technology, Chemnitz, Germany

Godehard Sutmann[*]
Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany
and ICAMS, Ruhr-University, Bochum, Germany

(Received 8 July 2013; published 19 December 2013)

Based on a parallel scalable library for Coulomb interactions in particle systems, a comparison between the fast multipole method (FMM), multigrid-based methods, fast Fourier transform (FFT)-based methods, and a Maxwell solver is provided for the case of three-dimensional periodic boundary conditions. These methods are directly compared with respect to complexity, scalability, performance, and accuracy. To ensure comparable conditions for all methods and to cover typical applications, we tested all methods on the same set of computers using identical benchmark systems. Our findings suggest that, depending on system size and desired accuracy, the FMM- and FFT-based methods are most efficient in performance and stability.

DOI: 10.1103/PhysRevE.88.063308 PACS number(s): 07.05.Tp, 41.20.Cv, 31.15.xv

I. INTRODUCTION

Particle simulation methods, like molecular dynamics or Monte Carlo sampling, are well-established numerical tools to understand the dynamics and structure of many-particle systems. Long-range interactions such as electrostatic or gravitational interactions pose a particular challenge to such simulations, since their computation is very time consuming. Simple truncation schemes for electrostatic interactions have been shown to produce artifacts [1]. Therefore, one has to take into account all pair interactions, leading to an unfavorable complexity of $O(N^2)$ (where N is the number of particles). A number of efficient algorithms, in particular for electrostatic interactions, have been devised to reduce this computational effort. Although these algorithms compute the same quantities, namely electrostatic forces and energies, they differ largely in their properties.

The problem is complicated even more by the fact that it is common to apply periodic boundary conditions in order to reduce boundary effects (especially in systems with only a few particles) and to mimic an infinite system. This creates the problem of an infinite number of system replicas, in which the Coulomb sum converges very poorly due to its long-range nature. When bulk systems are considered, one usually employs periodic boundary conditions in all three spatial dimensions, while simulations of thin films and surfaces or nanotubes require three-dimensional systems with only two or one periodic dimensions, respectively.

A traditional way to sum up the infinite terms under periodic boundary conditions is the Ewald summation method [2], which splits the total contribution into a short-range part and a long-range part. Summing up the short-range part in real space and the long-range part in Fourier space leads to a summation of two rapidly converging sums. Although this representation is exact, it contains infinite sums and therefore calls for error controlled approximations in order to be applicable in computer simulations. The parameters entering the Ewald sum, i.e., the range of the short-range part, the number of Fourier modes in the long-range part, and the splitting parameter which controls the relative weight of both terms, can be optimized in such a way that the overall performance of the Ewald

[*]g.sutmann@fz-juelich.de

summation is reduced to $O(N^{3/2})$. Taking into account the upper limits in the sums of short- and long-range contributions, approximations can be obtained with a controllable upper error threshold [3,4]. Other methods, like the Lekner-sum [5,6], the Ladd-sum [7,8], or the Sperb-sum [9,10], are based on similar principles. Note that Ewald-like methods also exist for systems with only one or two periodic boundary conditions [11–14].

Although the Ewald sum removes the quadratic complexity, the numerical effort is still too large for systems extending to several million particles or long time simulations. For this reason, alternative methods were developed with a strongly reduced complexity of $O(N \log N)$ or even optimal complexity of $O(N)$. They can be classified into splitting methods (SMs) and hierarchical methods (HMs). SMs have the same underlying idea as the Ewald summation method; i.e., they split the total electrostatic interaction into a short-range part and a long-range part by introducing a differentiable, localized function $\varphi(r)$ which splits the Coulomb term into overlapping short- and long-range contributions, $1/r = \varphi(r)/r + [1 - \varphi(r)]/r$. This corresponds to introducing a modified charge distribution, leading to a smooth potential energy surface in the long-range part and a singular term plus a smooth term in the short-range part, which, in total, reproduces the Coulomb potential.

Examples for fast methods of the SM type are extensions of the Ewald sum which evaluate the long-range Fourier space contribution on the basis of a fast Fourier transform (FFT), thereby providing an $O(N \log N)$ complexity. The drawback is the necessity of introducing an FFT mesh, onto which particle properties are mapped. Methods like the particle-particle-particle mesh (P^3M) [15], particle-mesh Ewald (PME) [16], or the smooth particle-mesh Ewald (SPME) [17] mainly differ in the way the particle properties are evaluated on the grid and transferred back to the particles [18]. For example, the popular SPME can easily be converted into the most accurate and versatile P^3M algorithm by changing the precomputed influence function [19]. Using a grid obviously introduces a spatial discretization, which causes an error, that can, however, be controlled and minimized using accurate error estimates [15,20].

Instead of relying on an FFT, multigrid methods discretize the Laplace operator and thereby recast this partial differential equation (PDE) into a linear system of equations that can be solved iteratively. Although the method's complexity is optimal, $O(N)$, the accuracy depends on the operator's discretization order. Furthermore, to obtain a mesh-independent convergence, a hierarchy of nested grids is employed. Interpolation of particle properties onto the grid and back therefrom is handled in the same spirit as for the FFT-based methods.

In contrast, HMs do not rely on modified charge distributions but evaluate the short-range part of the Coulomb energy by the direct particle-particle sum, while the long-range part is expanded into a multipole series, therefore effectively introducing pseudoparticles, located at the expansion centers. The transition from the short- to long-range description in HMs usually exhibits a discontinuity in the potential, which originates from the transition of the electrostatic sources from (point) charges to multipoles. This discontinuity is often considered to be responsible for a drift in energy and momentum of the system. However, the size of the discontinuity can be reduced to machine precision by controlling the number of multipoles in the expansion, thereby lifting the associated problems in momentum and energy conservation.

Examples for HMs are the Barnes-Hut tree method [21] and the fast multipole method [22]. One of the advantages of these algorithms is the mesh-free approach, which does not couple the accuracy of the approximation to an underlying grid resolution. The number of multipoles in the potential expansion as well as the depth of the hierarchical subdivision of space determines the accuracy. These characteristics could render such methods preferable in simulations of inhomogeneous systems, where mesh-based approaches, like FFT- or multigrid-based methods, that use the same mesh spacing everywhere, become very memory-intensive and slow. In recent years, HMs have been extended to simulations under periodic boundary conditions [23].

Another approach for computing Coulomb interactions efficiently has been proposed by Maggs [24] and adapted for molecular dynamics (MD) simulations by Pasichnyk [25]. In this algorithm, a simplified version of electrodynamics is simulated on a discretized lattice. This method, called Maxwell equations molecular dynamics (MEMD), is not widely adopted, but offers the important advantage of intrinsic data locality that originates from a grid-based solver for electrodynamics. This does not only provide a good base for parallelization but offers the possibility for spatially varying dielectric properties in the system and—for a constant particle density—scales as $O(N)$.

Although various methods for long-range interactions in periodic boundary conditions exist (see also several reviews or textbook material [26–31]), only a certain subset of them has entered into widely used molecular dynamics codes for scientific computing, e.g., P^3M and SPME in GROMACS [32], P^3M in LAMMPS [33], SPME in NAMD [34], and P^3M in ESPRESSO [35]. This fact might be related either to long standing and continuously improved implementations of selected methods and also the large effort needed to implement a new optimized method. The last two arguments certainly apply to the fast multipole method. One argument against adopting the FMM into molecular dynamics codes was based on the observation that, depending on the implementation, the method does not necessarily conserve momentum and energy in dynamical simulations due to the asymmetry in evaluating pair interactions between particles. However, it is also well known that mesh-based methods suffer from not conserving momentum or energy. These objections are considered in the present article.

Since the evaluation of the long-range interactions is the most time consuming part in the force loop of typical MD simulations, the most important requirement is efficiency. The parallelization of Coulomb solvers requires both a good single-core optimization and an efficient and scalable parallel implementation. The present article compares implementations of the fast multipole method, fast Fourier-based Ewald summations in the version of P^3M and P^2NFFT, multigrid-based methods, and a Maxwell local solver, all of which solve the interaction between charged particles in three-dimensional periodic boundary conditions. All of these methods are provided within a scalable parallel library, SCAFACOS (scalable fast Coulomb solvers) [30,36], which can be easily linked to existing particle programs. In addition to three-dimensional periodic methods, it also offers methods for other types of boundary conditions (open, one-dimensional-, two-dimensional-periodic), as well as a tree code specialized for strongly inhomogeneous systems.

In the scope of this article the different methods are compared in terms of efficiency, complexity, accuracy, and scalability. Common features and differences are outlined, which might help programmers and users to decide which method would be optimal for their specific problem. Although comparisons between different grid-based Ewald methods [18,37,38] as well as P^{3}M and FMM [39] or standard Ewald techniques and FMM [40] have been presented in the past, a detailed numerical comparison of an extended class of methods is still missing. To be informative it is vital for such a comparison that all methods are benchmarked for the same test systems, containing extreme and common particle distributions, as well as on the same set of hardware architectures. The present article not only compares the methods in terms of numerical complexity and efficiency but also considers scalability on parallel architectures.

Since all methods presented here are included in the SCAFACOS library, our results may directly, and without great effort, be used in the simulation programs of the users. Our discussion of scalability on large parallel architectures gives a good overview on which methods should be considered as viable candidates for large scale simulations. It is understood that such a comparison must be based on the actual implementations of the different methods, which were carried out by different programmers. Further developments and optimizations may change the relative performance of the different methods in the future.

The remainder of the paper is organized as follows. Section II provides a short description of the different methods. Section III describes the benchmark setup. Section IV discusses the stability of the methods in an MD simulation. Section V gives a performance comparison of the methods and in Section VI we summarize our findings and comparisons.

II. METHODS

Assume $N \in \mathbb{N}$ charged particles with charge $q_l \in \mathbb{R}$ at position $\boldsymbol{x}_l \in [0,1]^3$, $l = 1,\dots,N$. We are interested in the fast evaluation of the potential

$$\Phi(\boldsymbol{x}) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{\substack{l=1\\ \boldsymbol{x}_l^{\boldsymbol{0}} \neq \boldsymbol{x}}}^{N} q_l \frac{1}{\|\boldsymbol{x}-\boldsymbol{x}_l^{\boldsymbol{n}}\|_2}, \tag{1}$$

and field

$$\boldsymbol{E}(\boldsymbol{x}) = -\nabla\Phi(\boldsymbol{x}) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{\substack{l=1\\ \boldsymbol{x}_l^{\boldsymbol{0}} \neq \boldsymbol{x}}}^{N} q_l \frac{\boldsymbol{x}-\boldsymbol{x}_l^{\boldsymbol{n}}}{\|\boldsymbol{x}-\boldsymbol{x}_l^{\boldsymbol{n}}\|_2^3}. \tag{2}$$

Here, $\boldsymbol{x}_l^{\boldsymbol{n}} := \boldsymbol{x}_l + \boldsymbol{n}$, $\boldsymbol{n} \in \mathbb{Z}^3$ are the periodic particle images and $\|\boldsymbol{x}\|_2$ denotes the Euclidean norm in $\mathbb{R}^3$. The absence of prefactors in (1) and (2) corresponds to Gaussian units, i.e., $\frac{1}{4\pi\epsilon_0} := 1$.

Note that the summation over $\boldsymbol{n} \in \mathbb{Z}^3$ is only conditionally convergent, so that its value depends on the order of summation. Typically, one assumes summation in spherically ascending shells. Most puzzling is the fact that, in general, the summation result is not periodic in the particle coordinates despite the regular image grid. However, it can be shown [41,42] that the electrostatic potential can be written as the sum of a contribution that is periodic in the particle coordinates and a shape-dependent term that depends only on the total dipole moment of the innermost image $\boldsymbol{n} = \boldsymbol{0}$. The periodic contribution is often called the *intrinsic* contribution, since it can be seen as the solution of the Poisson equation under strict periodic boundary conditions. This is equivalent to so-called metallic boundary conditions, when one assumes a metallic medium that surrounds the growing summation sphere [43]. All results presented in the following are for the intrinsic solution.

A. Splitting methods

The computation of the electrostatic potential features two problems. On the one hand, $\frac{1}{\|\boldsymbol{x}-\boldsymbol{x}_l\|_2}$ is decaying very slowly, making direct summation very inefficient. On the other hand, it has a singularity, which makes it hard to apply many of the convergence accelerating theorems.

To overcome this, the electrostatic potential (1) at $\boldsymbol{x}$ can be split into a short-range contribution Φ^{sr} and a smooth long-range contribution Φ^{sm} by

$$\Phi(\boldsymbol{x}) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{\substack{l=1\\ \boldsymbol{x}_l^{\boldsymbol{0}} \neq \boldsymbol{x}}}^{N} q_l \int_{\mathbb{R}^3} \underbrace{\frac{\delta\big(\boldsymbol{y}-\boldsymbol{x}_l^{\boldsymbol{n}}\big) - \varphi_l^{\boldsymbol{n}}(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|_2}}_{\text{short-range}} + \underbrace{\frac{\varphi_l^{\boldsymbol{n}}(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|_2}}_{\text{smooth}} \, \mathrm{d}\boldsymbol{y}, \tag{3}$$

where the index $\boldsymbol{n}$ distinguishes positions among the periodic images, $\delta(\boldsymbol{x}_l^{\boldsymbol{n}} - \boldsymbol{y})$ is the δ distribution for the point charge at $\boldsymbol{x}_l^{\boldsymbol{n}}$, and $\varphi_l^{\boldsymbol{n}}(\boldsymbol{y})$ is a splitting function for the charge at $\boldsymbol{x}_l^{\boldsymbol{n}}$, as illustrated in Fig. 1. The splitting function is chosen conveniently such that it decays fast enough in both real space and reciprocal space, which makes it possible to derive fast converging expressions both for the short-range and the long-range parts of (3).

The smooth long-range part is described by the charge distribution resulting from the splitting function as

$$\rho^{\mathrm{sm}}(\boldsymbol{x}) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{l=1}^{N} q_l \varphi_l^{\boldsymbol{n}}(\boldsymbol{x}) \tag{4}$$

in the domain $[0,1]^3$ with periodic boundary conditions. Unlike the original charge distribution, a sum of δ distributions, ρ^{sm} is a smooth function, so that Fourier transforms or grid-based solvers can be applied to it in order to evaluate $\Phi^{\mathrm{sm}}(\boldsymbol{x})$.

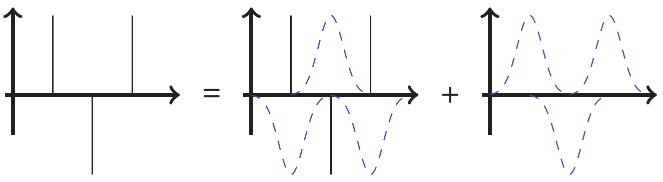

FIG. 1. (Color online) The charge distribution consisting of point charges (black bars) is split into a smooth part only (dashed blue lines) and the rest; compare with (3), taken from [44].

Note that the original sum (3) for the potential excludes the self-interaction $\boldsymbol{x} = \boldsymbol{x}_l$, while the charge distribution (4) sums over all charges, including possibly $\boldsymbol{x} = \boldsymbol{x}_j$, if we evaluate the potential or field at the position of a charge. Therefore, one has to subtract this self-contribution

$$\Phi^{\mathrm{self}}(\boldsymbol{x}_j) = q_j \int_{\mathbb{R}^3} \frac{\varphi_j^{\boldsymbol{0}}(\boldsymbol{y}+\boldsymbol{x}_j)}{\|\boldsymbol{y}\|_2} \mathrm{d}\boldsymbol{y}. \tag{5}$$

If the splitting function is radially symmetric around $\boldsymbol{x}_j$, this correction is only necessary for the potential, since the field contribution exactly cancels. If $\varphi_j^{\boldsymbol{n}}(\boldsymbol{y}+\boldsymbol{x}_j) = \varphi^{\boldsymbol{n}}(\boldsymbol{y})$, i.e., the splitting function arises by translation from a generic splitting function φ, then the self-contribution is equal for all charges and can be precomputed, often even analytically.

After some transformations, the short-range part results in

$$\Phi^{\mathrm{sr}}(\boldsymbol{x}) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{\substack{l=1\\ \boldsymbol{x}_l^{\boldsymbol{0}} \neq \boldsymbol{x}}}^{N} q_l \left[\frac{1}{\|\boldsymbol{x}_l^{\boldsymbol{n}} - \boldsymbol{x}\|_2} - \int_{\mathbb{R}^3} \frac{\varphi_l^{\boldsymbol{n}}(\boldsymbol{y})}{\|\boldsymbol{y}-\boldsymbol{x}\|_2} \mathrm{d}\boldsymbol{y} \right], \tag{6}$$

consisting of a direct summation part between the position $\boldsymbol{x}$ and all charges at $\boldsymbol{x}_l^{\boldsymbol{n}}$ in periodic images $\boldsymbol{n}$ and a correction term that removes the interaction with the charge distribution ρ^{sm}. If point-symmetric charge distributions $q_l\varphi_l^{\boldsymbol{n}}$ with limited support are chosen, the induced potential outside of this support is the same as that of a point charge beyond the support of the distribution. Therefore, the summands in (6) decay fast or vanish for particles that are far enough apart from each other that it can be evaluated by taking into account only the interactions up to a given cutoff distance r_{cut}. The rate of the decay depends on the splitting function.

The computation of the short-range component of the potential adheres to the same scheme as the computation of nonbonded short-range interactions in molecular dynamics. Various efficient algorithms and implementations with ideal scaling $O(N)$ exist. In this work, the short-range component was computed using a *linked cell* algorithm [28] that is implemented within the SCAFACOS library. In this algorithm, all particles are sorted into cells that are larger or equal to the short-range cutoff radius r_{cut}. To find all interaction partners of a particle in the short-range component, it is sufficient to compute the interactions with all particles in the neighboring cells, which yields the desired linear complexity. The parallel implementation employs a domain decomposition that distributes the particles uniformly among a Cartesian process grid. Particles at subdomain boundaries that are needed by more than one process are duplicated automatically during the particle redistribution step.

Note that in a molecular dynamics program, the short-range part of the SMs would typically be computed within the (possibly highly optimized) core of an MD program that computes other nonbonded short-range forces, so that the short-range computation of the library would not be used. This is expected to have a positive effect on the performance of the SMs when used in conjunction with an MD program; however, the actual gain depends on the employed MD program.

The remaining difficulty is to evaluate the long-range contribution by solving the Poisson equation

$$-\Delta\Phi^{\mathrm{sm}} = 4\pi\rho^{\mathrm{sm}} \tag{7}$$

subject to periodic boundary conditions on $[0,1]^3$. This solution can be obtained efficiently in a number of ways. In the set of presented methods we use either multigrid methods for the solution of the PDE- or Fourier-based methods.

1. *PDE-based: Multigrid*

The Poisson equation (7) is a prototypical elliptic PDE. All of its terms are discretized on a Cartesian grid of constant mesh size G_h of points

$$G_h = \{\boldsymbol{x} | \boldsymbol{x} = h\boldsymbol{j} \text{ for } \boldsymbol{j} \in \mathbb{Z}^3\},$$

with a formal discretization parameter $h \in \mathbb{R}$. Hence, the number of grid points per axis is $M = \frac{1}{h}$. Thereby, (7) becomes a linear system of equations,

$$\boldsymbol{A}_h u_h = f_h, \tag{8}$$

which is solved for the long-range potential Φ^{sm}, represented on the grid as u_h. A variety of discretizations $\boldsymbol{A}_h$ for the Laplace operator in (7) exist, each taking into account a specific number of neighboring grid points with appropriate coefficients used for evaluation, giving rise to a certain discretization order. For reasons of locality in implementations with emphasis on strong parallel scalability, a compact 27-point stencil of fourth order is typically used [45,46].

The smooth charge distribution ρ^{sm}, represented as f_h, is sampled at the grid points G_h to obtain the right-hand side. As splitting function, $\varphi(r)$, cardinal B splines are employed,

$$\varphi_l(\boldsymbol{x}) = \varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|_2),$$

that are radially symmetric. As an alternative, e.g., tensorized polynomials defined over intervals or Gaussians may be used.

Then, the solution of (8) can be obtained via an iterative relaxation scheme that minimizes the error $e_h^m = u_h - u_h^m$ of the discrete solution at iteration step m,

$$u_h^{m+1} = u_h^m + \boldsymbol{C}_h \underbrace{\left(f_h - \boldsymbol{A}_h u_h^m\right)}_{d_h^m}, \tag{9}$$

where $\boldsymbol{C}_h$ is an approximate inverse and d_h^m is the defect at iteration step m. Relaxation schemes differ on the choice of the approximate inverse $\boldsymbol{C}_h$ made clear when $\boldsymbol{A}_h = \boldsymbol{L}_h + \boldsymbol{D}_h + \boldsymbol{U}_h$ is split into a lower triangular matrix $\boldsymbol{L}_h$, diagonal matrix $\boldsymbol{D}_h$, and upper triangular matrix $\boldsymbol{U}_h$. For $\boldsymbol{C}_h := \boldsymbol{D}_h^{-1}$ we obtain the Jacobi method, for $\boldsymbol{C}_h := (\boldsymbol{L}_h + \boldsymbol{D}_h)^{-1}$ we have the Gauss-Seidel method. By introducing a weight ω in (9) that makes it possible to control the contribution of the defect to the update, relaxation methods are obtained.

It is well known that the spectral radius of the iteration matrix of both methods is bounded by 1. Furthermore, if the discretized operator $\boldsymbol{A}_h$ is represented by a compact stencil, then the component of the error e_h^m whose frequency is similar to the inverse length given by the discretization parameter h is decreased stronger than lower-frequency components. Hence, the iteration scheme converges more slowly the finer the grid is resolved.

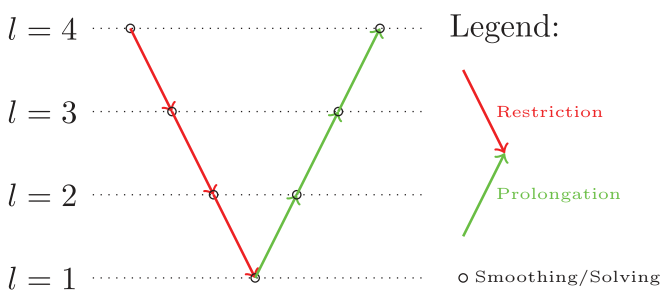

FIG. 2. (Color online) Depiction of typical V cycle.

However, if high-frequency components have been removed, the remaining error can be well described on a coarser grid. To this end, starting with a fine grid G_h we add a coarser grid G_H with commonly $h = \frac{H}{2}$. We define a restriction operator $I_h^H : G_h \to G_H$ and a prolongation operator $I_H^h : G_H \to G_h$, e.g., by trilinear interpolation. Then, the central idea of the multigrid method is to eliminate the high-frequency components of the defect, so-called smoothing, restrict the remaining error to a coarser grid, and solve the defect equation

$$A_H e_H^m = d_H^m, \tag{10}$$

where $d_H^m = I_h^H d_h^m$, on the coarser grid. The exact solution of (10) is the error e_H^m on the coarser grid that is prolongated back as a coarse-grid correction to the finer grid,

$$u_h^{m+1} = u_h^m - I_H^h A_H^{-1} I_h^H \big(A_h u_h^m - f_h\big). \tag{11}$$

This is commonly known as a two-grid cycle.

Although the defect equation (10) has to be solved exactly on the coarser grid, the resulting correction (11) is interpolated to the finer grid which allows for an approximate solution to suffice. Therefore, we may use again a two-grid cycle: This time we use the former coarse grid as the new fine grid, remove the high-frequency error, and apply a coarse-grid correction. This is especially favorable if the number of grid points is a power of 2. This nesting of multiple grids can be continued until the evaluation of the exact solution is efficient or trivial on the coarsest grid; see Fig. 2.

The proceeding of the multigrid-based methods can be subsumed as follows.

Charge assignment. The right-hand side of the Poisson equation (7) is constructed by sampling (4) with the respective splitting function on the finest grid.

Solving the Poisson equation. Using a multigrid method the approximate solution Φ^{sm} is computed. The accuracy is directly related to discretization parameter h on the finest grid and hence to the number of nested levels l.

Evaluation of the potentials. The calculated approximation Φ^{sm} is interpolated to the particle positions using Newton interpolation of sufficient degree, e.g., using a third-order polynomial for a fourth-order accurate solution. The potential is then corrected by subtracting the self-contribution Φ^{self} given by (5) and by adding the short-range part Φ^{sr} given by (6). As mentioned above, efficient methods for short-range interactions are used here.

Evaluation of the fields. The polynomials of the previous step are analytically differentiated to obtain the fields due to the calculated potential surface.

Parameters. The multigrid methods are influenced by the following parameters: size of the finest grid $h = 2^l$ determined by the number of nested levels l, diameter of the splitting function's compact support, the degree of the Newton interpolation, the discretization order for the Laplace operator, the specific cycle type, the fixed number of iteration steps used for smoothing, and the iteration threshold for the global defect. The grid size and the diameter of the support are the parameters that have by far the strongest impact on precision: For a given extent of the support, the obtained accuracy is strongly depending on h (in case of a fourth-order scheme the error is reduced by a factor of 16, if h is reduced by a factor of 2). Usually, the principal precision of the method is set with the size of the finest grid with respect to the discretization order and can be further tuned by increasing the diameter of the compact support of the charge distribution. Other parameters such as discretization order and interpolation degree also affect precision but the difference in speed is negligible.

The influence of the parameters on the computational work is described in the following. Initially, we have $O(m^3 N)$ work for the interpolation of the spline functions, where m is the number of grid points per axis of the compact support of the splitting function. Moreover, the computational work W_k per multigrid cycle for k levels is completely determined by its finest grid; see [47], Chap. 2.4.3]. Hence, assuming $W_l^{l-1} = O(M_l^3)$, where M_l is the number of grid points per dimension of level l, we obtain $O(M_k^3) + W_0$ computational work. The work on the coarsest grid W_0 is essentially constant, as the convergence of the multigrid is M-independent [47], Chap. 2.9.3]. The interpolation of the potential and the derivation of the interpolation polynomials for the calculation of the fields results in $O(l^3 N)$ work, where l is the interpolation degree. The necessary correction of the potential and the fields due to the short-range part is of order $O(N)$, if the particles' distribution is close to uniform. Hence, the method yields optimal complexity with the scaling constant depending critically on the desired accuracy determined by the discretization order of the stencil.

Multigrid is usually parallelized by a suitable decomposition of the domain and distribution of particles over many processes. Each process samples its local share of particles on the grid and performs operations on its local grid only, where some communication is required with direct neighboring processes during the restriction, prolongation, and smoothing. There is also one global communication in every multigrid iteration where the local defects are summed up globally. Eventually, each process interpolates back its particle potential.

2. *Fourier-based: Ewald and particle-mesh Ewald*

The fundamental idea of the Fourier-based methods is to compute the smooth long-range contribution Φ^{sm} in Fourier space. This leads to the well-known Ewald formula [2,48] for the computation of (1), which splits the electrostatic potential at position $\boldsymbol{x}_j$ into the parts $\Phi(\boldsymbol{x}_j) \approx \Phi^{\text{sr}}(\boldsymbol{x}_j) + \Phi^{\text{sm}}(\boldsymbol{x}_j) + \Phi^{\text{self}}(\boldsymbol{x}_j)$, where

$$\Phi^{\text{sr}}(\boldsymbol{x}_j) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3} \sum_{\substack{l=1 \\ \boldsymbol{x}_l^{\boldsymbol{0}} \neq \boldsymbol{x}_j}}^{N} q_l \frac{\operatorname{erfc}\big(\alpha \|\boldsymbol{x}_j - \boldsymbol{x}_l^{\boldsymbol{n}}\|_2\big)}{\|\boldsymbol{x}_j - \boldsymbol{x}_l^{\boldsymbol{n}}\|_2}, \tag{12}$$

$$\Phi^{\mathrm{sm}}(\boldsymbol{x}_j) = \sum_{\boldsymbol{k}\in\mathcal{I}_M\setminus\{\boldsymbol{0}\}} \frac{\mathrm{e}^{-\pi^2\|\boldsymbol{k}\|_2^2/\alpha^2}}{\pi\,\|\boldsymbol{k}\|_2^2} S_{\boldsymbol{k}}\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}_j}, \qquad \Phi^{\mathrm{self}}(\boldsymbol{x}_j) = -2q_j\frac{\alpha}{\sqrt{\pi}}. \tag{13}$$

Hereby, the complementary error function is defined by $\mathrm{erfc}(z) = \frac{2}{\sqrt{\pi}}\int_z^\infty \mathrm{e}^{-t^2}\mathrm{d}t$ and the structure factors $S_{\boldsymbol{k}}$ are given by

$$S_{\boldsymbol{k}} = \sum_{l=1}^{N} q_l \mathrm{e}^{+2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}_l},$$

where M is the number of grid points per dimension and the multi-index set $\mathcal{I}_M := \{-\frac{M}{2},\ldots,\frac{M}{2}-1\}^3$ collects all the grid points. The short-range part (12) as well as the Fourier coefficients in (13) decay exponentially fast, so that the potentials $\Phi(\boldsymbol{x}_j)$, $j=1,\ldots,N$, can be computed in $O(N^{3/2})$ when choosing optimal parameters [49].

The evaluation of the smooth component Φ^{sm} can be further sped up using FFTs, which leads to the family of *PME* algorithms that includes P^{3}M [15], PME [16], SPME [17], and P^{2}NFFT [50]. If the charge positions $\boldsymbol{x}_j$ are "sufficiently uniformly distributed," these algorithms end up with an arithmetical complexity of $O(N\log N)$.

During the end of the 1990s, it became clear that P^{3}M, PME, and SPME can be considered as a single method with different components. These are mostly interchangeable, and their choice has a significant impact on the performance of the method [18,51]. The original P^{3}M algorithm by Hockney and Eastwood has the advantage that it replaces the continuum Green's function by what is called the *optimal influence function*, which can be derived analytically by minimizing the functional that yields an average root mean square error for the force. The optimal influence function can be derived for many other variants that minimize the error in energy, or for dipolar forces, or for the interlacing technique described further below [15,52–54]. The method P^{3}M in the present article refers to the best known combination of these components, which means that it differs from the original method by Hockney and Eastwood [15] in several details.

The approximate computation of the long-range components consists of the following steps.

Charge assignment. The charges are smeared out onto the P nearest grid points of a discrete grid ρ_{mesh} of size M^3,

$$\rho_{\mathrm{mesh}}(\boldsymbol{p}) = \frac{1}{h^3}\sum_{l=1}^{N} q_l\varphi(h\,\boldsymbol{p}-\boldsymbol{x}_l),$$

for $\boldsymbol{p}\in\mathcal{I}_M$, where $h=1/M$ is the grid spacing. In P^{3}M, the window function φ is chosen as a three-dimensional tensor product of cardinal B splines of order P.

Forward Fourier transform. The charge grid ρ_{mesh} is Fourier transformed using the FFT to yield the reciprocal charge distribution $\hat{\rho}_{\mathrm{mesh}}$.

Solving for the potential. Next, the reciprocal potential $\hat{\Phi}^{\mathrm{sm}}$ is computed from $\hat{\rho}_{\mathrm{mesh}}$ using an appropriate Green's function $\hat{G}_{\mathrm{opt}}$ (which is often called *influence function* in this context):

$$\hat{\Phi}^{\mathrm{sm}} = \hat{G}_{\mathrm{opt}}\hat{\rho}_{\mathrm{mesh}}.$$

Note that $\hat{G}_{\mathrm{opt}}$ is, in fact, the product of the Coulomb Green's function and the Fourier transform of the Gaussians used for the Ewald splitting. In continuum, this function is given by

$$\hat{G}(\boldsymbol{k}) = \frac{1}{\pi\,\|\boldsymbol{k}\|_2^2}\mathrm{e}^{-\pi^2\|\boldsymbol{k}\|_2^2/\alpha^2}.$$

However, since the charge density is discrete, this continuum Green's function is not the best choice, even if it has been used by some other methods. To minimize the overall relative error in the potential, for example, the *optimal influence function* is given by [52]

$$\hat{G}_{\mathrm{opt}}(\boldsymbol{k}) = \frac{\sum_{\boldsymbol{m}\in\mathbb{Z}^3}\hat{\varphi}^2(\boldsymbol{k}+\boldsymbol{m})\hat{G}(\boldsymbol{k}+\boldsymbol{m})}{\left[\sum_{\boldsymbol{m}\in\mathbb{Z}^3}\hat{\varphi}^2(\boldsymbol{k}+\boldsymbol{m})\right]^2},$$

where the sum over $\boldsymbol{m}$ is known as the *aliasing* sum and serves to minimize the discretization errors.

Backward Fourier transform. Using the FFT, the reciprocal potential is Fourier transformed backward to yield the long-range potential $\Phi^{\mathrm{sm}}_{\mathrm{mesh}}$ at the grid points.

Evaluation of the potentials. The potentials at the original particle positions are approximated from the potential grid. This step employs the same window function as the charge assignment,

$$\Phi^{\mathrm{sm}}(\boldsymbol{x}) = \sum_{\boldsymbol{p}\in\mathcal{I}_M}\Phi^{\mathrm{sm}}_{\mathrm{mesh}}(\boldsymbol{p})\varphi(\boldsymbol{x}-h\,\boldsymbol{p}).$$

Evaluation of the fields. For the P^{3}M method in the present paper, the fields are derived with an *analytical differentiation* scheme if not otherwise stated. This method is identical to the approach used in the SPME method [17]. Since the gradient of the charge assignment function is known analytically, the gradient at the original particles positions can be directly interpolated from the values of the potential grid,

$$\boldsymbol{E}^{\mathrm{sm}}(\boldsymbol{x}) = \sum_{\boldsymbol{p}\in\mathcal{I}_M}\Phi^{\mathrm{sm}}_{\mathrm{mesh}}(\boldsymbol{p})\nabla_{\boldsymbol{x}}\varphi(\boldsymbol{x}-h\,\boldsymbol{p}).$$

A second approach, which was, for example, employed by Darden *et al.* in their PME variant [16], is the i$\boldsymbol{k}$ *differentiation*. Here, one makes use of the fact that in reciprocal space, the spatial derivative turns into a simple multiplication by i$\boldsymbol{k}$, where $\boldsymbol{k}$ denotes the wave vector. This makes it computationally very cheap to compute the reciprocal electric field $\hat{\boldsymbol{E}} = \mathrm{i}\boldsymbol{k}\hat{\Phi}^{\mathrm{sm}}$. However, transforming this field back to real space requires three backward FFTs instead of only one that is sufficient for the scalar potential.

Interlacing. In addition, the P^{3}M algorithm in this work uses an extension to the algorithm sketched above that is called *interlacing* [54], and that was already suggested by Hockney and Eastwood [15]. A second grid shifted by half a grid spacing is introduced and all the steps above are applied to both grids. Afterward, the potentials and fields obtained for both grids are averaged. This results in about an order of magnitude higher accuracy compared to the single grid method, while the computational effort for using interlacing is roughly twice the effort of using a single grid. The gain in accuracy is sufficient to make it possible to reduce the mesh size by factor of two in all three spatial dimensions while maintaining the accuracy, so that interlacing yields a significant performance gain.

Throughout the rest of this work, we use the term P^{3}M to denote the variant of the algorithm with the best combination of components known to date, namely the optimal influence function, interlacing, and analytical differentiation [51].

Parameters. The parameters of the method are the Ewald splitting parameter α, the short-range cutoff radius r_{cut}, the order of the charge assignment function P, and the grid spacing h. The choice of these parameters does have a strong influence on the accuracy and performance of the algorithm. Fortunately, good analytical error estimates exist [20] that help to make a good choice. The Ewald splitting parameter α does not have a direct influence on the computational performance; however, it does have a strong influence on the accuracy of the algorithm. There is an "optimal" value of α where the best accuracy is reached. Using a value that is off this optimum by as little as 10% can result in an accuracy that is two orders of magnitude worse. Using the analytical error estimates, the optimal α can be determined easily. The short-range cutoff radius r_{cut} determines the accuracy of the short-range part, while the charge assignment order P and grid spacing h determine the accuracy of the long-range part. Increasing the first two and decreasing the latter will improve the accuracy of the methods at the expense of the performance. Determining the optimal combination of these parameters to reach a given accuracy is an optimization problem.

The largest part of the parallelization of the algorithm is straightforward. The most complex part is the parallelization of the three-dimensional FFT, which was parallelized using two-dimensional stencils [55].

The particle-particle NFFT (P^{2}NFFT) is a general framework for particle mesh algorithms based on non-equispaced fast Fourier transforms (NFFTs) [56,57]. By appropriate choice of parameters, this framework includes the PME methods for periodic boundary conditions and the fast summation algorithm [58,59] for nonperiodic boundary conditions.

In the case of periodic boundary conditions, the P^{2}NFFT follows the approach of [60,61], e.g., the NFFT is applied for the fast calculation of the long-range parts Φ_j^{sm}. The structure factors $S_{\boldsymbol{k}}$ in (13) can be computed by an adjoint, three-dimensional NFFT of total grid size M^3 with $O(N + M^3 \log M)$ arithmetic operations. This is followed by M^3 multiplications in Fourier space and completed by a three-dimensional NFFT of total grid size M^3 to compute the outer sums. The relation between N and M is determined by the approximation error of the algorithm and is discussed in detail in [18,20,57]. Choosing the total grid size M^3 proportional to the number of particles N yields the typical overall complexity of $O(N \log N)$.

The modularized structure of the P^{2}NFFT allows a straightforward parallelization based on the parallel NFFT algorithms presented in [50], which are implemented within the publicly available PNFFT software library [62]. A regular blockwise domain decomposition is induced by the underlying parallel FFT algorithms [63] that are publicly available within the PFFT software library [64].

Although different in spirit, the steps of P^{3}M and P^{2}NFFT are very similar, and in fact, it can be shown that these two methods are equivalent for periodic boundary conditions [65]. The optimal influence function of P^{3}M, which is the result of a functional optimization, in the light of the NFFT algorithm is nothing but the continuum Green's function decorated by the convolution and deconvolution steps of the NFFT and its adjoint. This analogy makes it possible to employ interlacing also to the NFFT-based algorithm and gives deeper insight into the origin of P^{3}M's optimal influence function. Also, the computation of the gradients is handled similarly by both methods.

Both methods P^{3}M and P^{2}NFFT use an equal distribution of the particle system among a Cartesian process grid, thus requiring a redistribution of the particle data. If particles are already located on their corresponding process of the grid, then the amount of communication required for the redistribution decreases automatically. This helps to improve the scalability of the parallel implementations, especially when the number of particles per process becomes very low.

B. Hierarchical methods

In comparison to Fourier-based methods, HMs like the Barnes-Hut tree method [21] or the fast multipole method (FMM) [66] do not carry out any computations in reciprocal space. Instead, the fast evaluation of the potentials $\Phi(\boldsymbol{x}_j)$ is achieved by splitting up contributions in real space into a near-field part and a far-field part. The reduction of numerical complexity is accomplished by factorization of the inverse distance $\|\boldsymbol{x}_j - \boldsymbol{x}_l\|_2^{-1}$ into parts which only depend on $\boldsymbol{x}_j$ and parts which only depend on $\boldsymbol{x}_l$. The expansion can be performed either in Cartesian or, more efficiently, spherical coordinates [67]. Interactions in tree codes consider contributions between particles and a whole hierarchy of pseudoparticles, consisting of multipoles located at expansion centers, and therefore result in a complexity of $O(N \log N)$. Often a geometric criterion is considered to decide about the size of a pseudoparticle, i.e., the spatial volume in which explicit particles are grouped together into a multipole expansion, interacting with a single particle in a distance. On the other hand, FMM makes use of hierarchically transferring multipole information down a tree, so that individual particles interact with effective far-field and explicit near-field particles, which gives rise to an $O(N)$ complexity. In the following the FMM is described on a more detailed level.

1. *Fast multipole method*

With the help of the associated Legendre polynomials P_{lm} and a transformation of the particle coordinates into a spherical representation $\boldsymbol{x}_l = \boldsymbol{a} = (a,\alpha,\beta)$, $\boldsymbol{x}_j = \boldsymbol{r} = (r,\theta,\phi)$ a single particle-particle interaction can be factorized for $\|\boldsymbol{a}\|_2 < \|\boldsymbol{r}\|_2$ via

$$\begin{aligned}\frac{1}{\|\boldsymbol{r}-\boldsymbol{a}\|_2} &= \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\frac{(l-m)!}{(l+m)!}\frac{a^l}{r^{l+1}}\\ &\quad\times P_{lm}(\cos\alpha)P_{lm}(\cos\theta)\mathrm{e}^{-\mathrm{i}m(\beta-\phi)}\\ &= \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\omega_{lm}(\boldsymbol{a})\mu_{lm}(\boldsymbol{r}).\end{aligned}$$

For numerical reasons it is sufficient to truncate the infinite series at a certain finite term p for the computation. Besides the truncation at multipole p, the FMM has two additional parameters. To establish a spatial grouping of particles and to apply the factorization scheme, HMs employ a decomposition of space into a set of boxes eventually forming an octree. The subdivision is repeated until a certain tree depth $d_{\max}$ is reached. The last parameter, the "well separatedness" (s), controls the convergence rate of the aforementioned expansion. The minimum separation of two boxes interacting via multipoles is $s = 1$. Higher separation yields better convergence, thus lowering the number of poles p for a given accuracy, but increases the size of the interaction set in both near and far field. To compute the potentials $\Phi(\boldsymbol{x}_j)$ and fields $\boldsymbol{E}(\boldsymbol{x}_j)$, the following steps have to be implemented.

Expansion into multipoles. First, all particles need to be sorted into their corresponding boxes of the octree at depth $d_{\max}$. This step is performed via a Radix sort method [68]. The sorting is followed by an expansion of all M_k particles around their box center into spherical multipoles ω_{lm}^k on the lowest level $d_{\max}$ of the tree via

$$\omega_{lm}^k = \sum_{j=1}^{M_k} q_j a_j^l P_{lm}(\cos\alpha_j) \mathrm{e}^{-\mathrm{i}m\beta_j}.$$

Since all M_k particles inside each box k are expanded around the same center, the coefficients of the particles can be summed up into a unique expansion ω_{lm}^k per box. This step takes $O(p^2 N)$ time.

Shifting multipoles. Now the multipole information is shifted from the lowest level upwards to the root node of the tree via the multipole-to-multipole operator ($\mathcal{O}_{M\to M}$)

$$\omega_{lm} = \sum_{j=0}^{l} \sum_{k=-j}^{j} \mathcal{O}_{M\to M}(\boldsymbol{b})\omega_{jk}(-\boldsymbol{b}).$$

The $\mathcal{O}_{M\to M}$ operator combines multipole coefficients from up to eight boxes on depth d and transforms these coefficients into a single multipole expansion ω_{lm} around the center of the parent box $\boldsymbol{b}$ at depth $d-1$. This step is repeated for each level in the tree until the root node is reached. For a homogeneous particle distribution the shift can be performed in $O(p^4 N)$ time.

Far-field interactions. After the octree has the full multipole information available in each box on each level, the far-field interactions can be performed. To obtain a reduced complexity, only a fixed number of close-by interactions per level are taken into account. Omitted interactions are carried out on a higher level of the tree. In a tree code M_k remote particles of a box k are transformed into local coefficients μ_{lm} around the center of the box under consideration with the help of the particle-to-local operator (P2L) via

$$\mu_{lm} = \sum_{j=1}^{M_k} q_j \frac{1}{r_j^{l+1}} P_{lm}(\cos\theta_j) \mathrm{e}^{\mathrm{i}m\phi_j}.$$

Since each particle on each level has to be taken into account this step costs $O(p^2 N \log N)$ time.

The complexity can be reduced to $O(p^4 N)$ in the FMM by applying the multipole-to-local operator ($\mathcal{O}_{M\to L}$) given by

$$\mu_{lm} = \sum_{j=0}^{p} \sum_{k=-j}^{j} \mathcal{O}_{M\to L}(-\boldsymbol{b})\omega_{jk}(\boldsymbol{b}).$$

The reduced complexity originates from the possibility to transform multipoles of a certain box directly into a local expansion without the need to incorporate the particles itself. The number of interactions for the FMM is limited to $7(2s+1)^3$, since only boxes of s neighboring parent boxes on level $l-1$ are considered. For $s = 1$ the maximum number of interactions for each box on each level is 189. A higher separation criterion increases the interaction list substantially and is not favorable for the accuracy range discussed in this article.

Shifting local coefficients. Now the far-field information is available as local coefficients μ_{lm} inside the tree. To compute the far-field properties (e.g., potentials, fields) these coefficients are shifted towards the leaves of the tree. This step uses the local-to-local operator ($\mathcal{O}_{L\to L}$) given by

$$\mu_{lm} = \sum_{j=0}^{p} \sum_{k=-j}^{j} \mathcal{O}_{L\to L}(\boldsymbol{b})\mu_{jk}(-\boldsymbol{b}).$$

This step can be performed in $O(p^4 N)$ time.

Compute far field. The far-field contributions now can be computed for each particle in each box. The far-field approximation of the potential $\Phi(\boldsymbol{a})$ can be obtained by

$$\Phi(\boldsymbol{a}) \approx \sum_{l=0}^{p} \sum_{m=-l}^{l} \mu_{lm} \frac{1}{(l+m)!} a^l P_{lm}(\cos\alpha) \mathrm{e}^{-\mathrm{i}m\beta}.$$

This step can be performed in $O(p^2 N)$ time.

Compute near field. Due to the convergence requirement $\|\boldsymbol{a}\|_2 < \|\boldsymbol{r}\|_2$ some interactions may not have been accounted for. These remaining near-field interactions are carried out separately with a classical direct summation scheme. Since the number of particles in the near field is bounded, this step also scales linearly.

To allow faster high-precision calculations, the operator complexity of the current FMM implementation uses a rotation-based approach [69], reducing the complexity from $O(p^4 N)$ to $O(p^3 N)$ without inflicting the error bounds. The presented implementation also eliminates the need to seek for the optimal set of FMM parameters by utilizing an error control and run time minimization scheme [70] based on a user-provided energy error threshold ΔE. The algorithm can also handle (mixed) periodic boundary conditions [23,30] efficiently in $O(N)$ time.

Parameters. The parameters which determine the accuracy and performance of the FMM are, as mentioned above, the well-separated criterion s, the depth of the FMM tree d, and the length of the multipole expansion p. All three parameters are determined by an error prediction model, for which an optimization problem is solved,

$$\frac{\partial t}{\partial d} = 0, \quad \frac{\partial t}{\partial p} = 0, \quad \frac{\partial t}{\partial s} = 0,$$
$$\text{subject to} \quad \Delta E(d,p,s) \leqslant \epsilon,$$

which determines the parameters according to the threshold and a minimum run time. Therefore, the only parameter, which has to be specified externally is the error threshold ϵ.

The FMM implementation uses parallel sorting to insert particles into their corresponding boxes of the octree and to distribute these boxes among parallel processes. The resulting

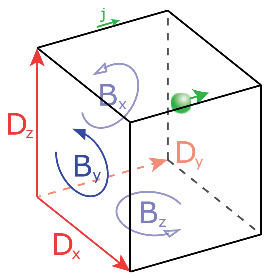

FIG. 3. (Color online) Schematic of the MEMD lattice interpolation. The electric fields $\boldsymbol{D}$ are placed on lattice sites and the magnetic fields $\boldsymbol{B}$ in rotated dual space on the plaquettes. The current $\boldsymbol{j}$ is interpolated from the moving charges.

distribution of particles among processes corresponds to a Z-order space filling curve. If particles are already provided with this kind of distribution, then the amount of communication required for the parallel sorting decreases automatically, thus improving the scalability of the parallel implementation.

C. Local method: MEMD

While the majority of electrostatics algorithms calculate Coulomb interactions by computing the global potential, for example by solving the Poisson equation, and differentiating the resulting potential, the MEMD algorithm follows a different approach. It is based on the full electrodynamics of the system, discretized on a lattice as shown in Fig. 3.

Initial solution and temporal updates. The method consists of two different combined methods. Initially, an exact solution of the Gauss equation

$$\nabla \cdot \boldsymbol{D} = \rho$$

for the system is computed with a numerical relaxation scheme. The charges are interpolated onto the lattice via the linear cloud-in-cell algorithm. They are then added up shellwise to perfectly obey Gauss' law, and the field energies are minimized numerically. This initial scheme scales with $O(N^2)$, but has to be applied only once in the beginning and still retains the locality of the algorithm. A possible improvement methods with better scaling behavior shall be discussed elsewhere.

Subsequently, the correct solution can be obtained by only applying temporal updates of the fields. The time derivative and some physical arguments, as laid out in [25], lead to the following constraint that is then applied to the propagation of the system

$$\frac{\partial}{\partial t}\boldsymbol{D} + \boldsymbol{j} - \frac{1}{c^2}\nabla \times \boldsymbol{B} = 0,$$

with the electric field $\boldsymbol{D}(\boldsymbol{x}) = \varepsilon(\boldsymbol{x})\boldsymbol{E}(\boldsymbol{x})$ [with the assumption of a local $\varepsilon(\boldsymbol{x})$], the electric current $\boldsymbol{j}$ and a magnetic field component $\boldsymbol{B}$. This results in temporal updates for the electric field without ever calculating the corresponding potential $\Phi(\boldsymbol{x})$. The equations are purely local; hence, the permittivity $\varepsilon(\boldsymbol{x})$ is not necessarily constant but may vary in space.

Thermodynamic limit. Because of the algorithm's locality, we consider the Lagrange density function for this constraint

$$\mathcal{L} = \sum_i \frac{m_i}{2}\boldsymbol{v}_i^2 - U + \frac{1}{2c^2}\int \varepsilon(\boldsymbol{x})\dot{\boldsymbol{\Theta}}^2 \mathrm{d}\boldsymbol{x} - \frac{1}{2}\int \frac{\boldsymbol{D}^2}{\varepsilon(\boldsymbol{x})}\mathrm{d}\boldsymbol{x} + \int \boldsymbol{A}(\dot{\boldsymbol{D}} - \nabla \times \dot{\boldsymbol{\Theta}} + \boldsymbol{j})\mathrm{d}\boldsymbol{x},$$

where m_i and $\boldsymbol{v}_i$ are the particle masses and velocities, $\boldsymbol{\Theta}$ is an additional degree of freedom in form of a vector field that relates to the magnetic field, $\boldsymbol{A}$ is a Lagrange multiplier, and $1/c^2$ is merely a prefactor that can be expressed by the physical wave propagation speed c.

Equations of motion. From this, the equations of motion for the particles and the fields are obtained via variational calculus. For the particles, this yields the known formula for the Lorentz force, and the artificial $\boldsymbol{B}$ field shows the wavelike propagation

$$\begin{aligned} m_i\ddot{\boldsymbol{x}}_i &= -\frac{\partial U}{\partial \boldsymbol{x}_i} - q_i\boldsymbol{E} + q_i\boldsymbol{v}_i \times \boldsymbol{B}, \quad \boldsymbol{B} = \frac{1}{c^2}\dot{\boldsymbol{\Theta}}, \\ \dot{\boldsymbol{D}} &= c^2\,\nabla \times (\nabla \times \boldsymbol{B}) - \frac{\boldsymbol{j}}{\varepsilon}, \quad \dot{\boldsymbol{B}} = -\nabla \times \boldsymbol{D}. \end{aligned} \tag{14}$$

Note that the magnetic part of the Lorentz force can and should be omitted since the $\boldsymbol{B}$ field is artificial and only used to propagate changes in the $\boldsymbol{D}$ field. To preserve time reversibility of the external integrator, the magnetic fields are propagated twice by half a time step, before and after the force calculation, respectively.

The intrinsic locality of the algorithm leads to an $O(N)$ scaling and a way to treat periodic box geometries by matching the box boundaries onto a torus.

Implementation. For parallel execution, the system is split into cubic spatial domains and distributed on the available cores. Each domain contains a regular lattice that carries all currents and fields. Since the algorithm is purely local, communication only occurs on the lattice cubes directly attached to a neighbor domain. These surface patches are exchanged asynchronously while the $\boldsymbol{B}$ fields are propagated for all inner cells during parallel communication. The communication cost can be reduced if the charges are presorted in cubic domain decomposition. Especially for small numbers of particles per core, where the percentage of computing time for the sorting step is significant.

Parameters. The algorithm features one central parameter, the lattice size, to tune speed and accuracy. All other parameters are either given by the external integration routine or are constrained by the stability criterion. They can thus be set optimally by the implementation. Optimal accuracy is achieved in an error minimum for an appropriate mesh. The accuracy reacts sensitively to coarser mesh sizes, with a scaling of a^3 with the lattice spacing a. Finer meshes, scaling with $1/a^2$, do not influence the accuracy as strongly but increase the computational effort proportional to $1/a^3$ [71]. The choice of this parameter can therefore influence the behavior of the algorithm significantly.

III. BENCHMARK SETUP

A. Systems

In order to calculate the electric field $\boldsymbol{E} = -\nabla\Phi$ for a $1/r$ potential, all methods presented here split the long-range

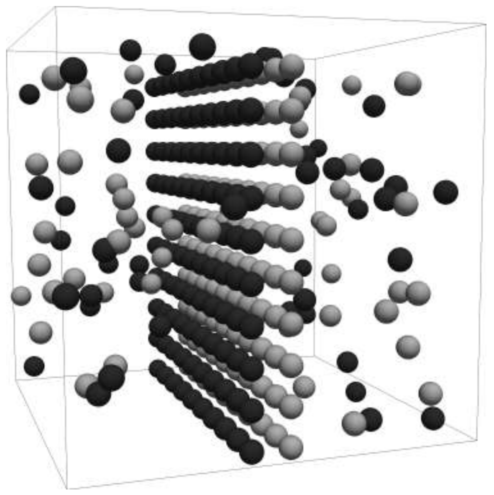

FIG. 4. The cloud-wall system (300 charges): two oppositely charged walls in the center of the box and a surrounding diffuse cloud. The system was artificially created to contain a strong long-range field component.

part of the potential from the divergence at $r \to 0$. Since the performance relevant part of the calculations is the treatment of the long-range components, all benchmarks presented here were carried out for two systems, consisting of different charge distributions, both of which feature a significant long-range contribution which challenges the achievable accuracy of the methods.

The cloud-wall model system, shown in Fig. 4, consists of 300 particles, which represent two oppositely charged walls centered in a cubic box together with a diffuse cloud of charges. This ensures a strong long-range contribution in the potential. The periodic box was replicated 3, 7, 15, 32, and 70 times in every direction to yield cubic boxes filled with 8100, 102 900, 1 012 500, 9 830 400, and 102 900 000 particles, respectively.

The cloud-wall systems were used for both the performance measurements as function of accuracy in Sec. V A as well as for the scalability benchmarks in Sec. V C. Since these test cases represent periodically replicated systems, the reference values for potentials and forces can be obtained even for very large numbers of particles.

The second test system consists of a cubic box filled with 12 960 particles of a silica melt shown in Fig. 5. It was taken from an MD simulation of a melting silica crystal using the Beest–Kramer–van Santen (BKS) force field [72]. The overall charge neutral system consists of positively and negatively charged ions which are sufficiently homogeneously distributed, while the electrostatic potential still has a significant long-range contribution. For the scaling and benchmark runs the original silica melt system was replicated 2, 4, 8, 16, and 32 times in every direction to yield cubic boxes filled with 103 680, 829 440, 6 635 520, 53 084 160, and 424 673 280 particles, respectively.

The silica melt test systems were used for both the stability benchmarks in Sec. IV and the complexity benchmarks in Sec. V B.

B. Error measure

In order to compare the accuracy of the different methods, the following error measure is defined. Let $\Phi_{\mathrm{FCS}}(\boldsymbol{x}_j)$ denote

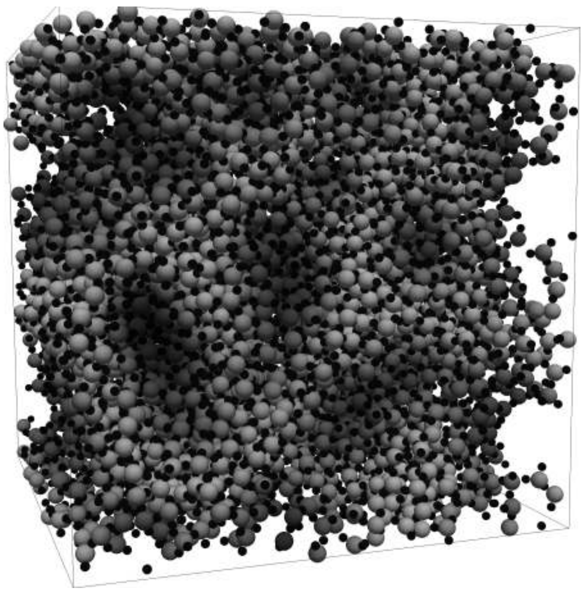

FIG. 5. Silica melt (12 960 charges): a system that is sufficiently homogeneous while retaining a significant long-range contribution.

the potential which is calculated by one of the presented fast Coulomb solvers and $\Phi_{\mathrm{REF}}(\boldsymbol{x}_j)$ the highly accurate reference potential computed by the Ewald summation method for which the parameters were chosen to yield an accuracy close to machine precision. In the following, we compare the different solvers with respect to the relative rms potential error given by

$$\varepsilon_{\mathrm{pot}} := \left(\frac{\sum_{j=1}^{N} |\Phi_{\mathrm{REF}}(\boldsymbol{x}_j) - \Phi_{\mathrm{FCS}}(\boldsymbol{x}_j)|^2}{\sum_{j=1}^{N} |\Phi_{\mathrm{REF}}(\boldsymbol{x}_j)|^2} \right)^{1/2} .$$

Since all methods differ in their definition of relative short- and long-range contribution, only the total potential can be used as common reference point.

C. Architectures

The benchmark tests were performed on two different hardware architectures at Jülich Supercomputing Centre. In the meantime, the JUGENE architecture has been shut down and replaced by the JUQUEEN system [73].

1. *Blue Gene/P (JUGENE)* [74]. One node of a Blue Gene/P consists of four IBM PowerPC 450 cores that run at 850 MHz. These four cores share 2 GB of main memory. Therefore, we have 0.5 GB RAM per core, whenever all the cores per node are used. The nodes are connected by a 3d-torus network with 425 MB/s bandwidth per link. In total JUGENE consists of 73 728 nodes, i.e., 294 912 cores. The software has been built with the IBM XL compilers (Advanced Edition for Blue Gene/P, V9.0).

In this work, we consider this architecture as prototypical for a well interconnected HPC machine.

2. *Jülich research on Petaflop architectures (JUROPA)* [75]. One node of JUROPA consists of 2 Intel Xeon X5570 (Nehalem-EP) quad-core processors that run at 2.93 GHz. These eight cores share 24 GB DDR3 main memory. Therefore, we have 3 GB RAM per core, whenever all the cores per node are used. The nodes are connected by a QDR InfiniBand

network with nonblocking fat tree topology. In total, JUROPA consists of 2208 nodes, i.e., 17 664 cores. The software has been built with the Intel Compilers (version 11.1).

In this work, we consider this architecture as prototypical for a convenience cluster.

D. Implementations

Whenever the performance and, in particular, the scalability of an algorithm is examined, it strongly depends on the actual implementation of the algorithm. Therefore, in some cases different implementations of the same algorithm are employed, as provided within the scalable parallel library SCAFACOS [30,36], which is used for the comparison of methods.

In the following some implementation features are provided for each method. A more in-depth discussion about parallel implementations is given in the library manual [76] and in a follow-up publication that is still in preparation.

Two implementations of the multigrid method were used in the benchmarks. PP3MG [77] is implemented in C, featuring as splitting function either cardinal B splines or polynomials defined over an interval. Finite difference or finite volume operators are available as fourth-order compact schemes or extended higher-order schemes. Newton interpolation is applied to map grid values to particles. Versatile multigrid (VMG) is a multigrid-based method implemented in C++ with strong emphasis on modularity in terms of employed iterative solver, domain decomposition, and interpolation schemes. Note that both multigrid-based methods have converged to employing cardinal B splines or polynomials, a fourth-order compact stencil, and Newton interpolation in most of the following benchmarks and hence differences in performance can be recast to specific implementations and differences in the compilers. For the very low- and high-accuracy benchmarks lower or higher-order schemes are used.

In the case of the Fourier-based methods, two implementations were used which were originally developed in different communities. The P^3M algorithm or one of its variants is a widely used method in condensed matter simulations to compute Coulomb interactions. The implementation of the algorithm used in this work was originally adopted from the simulation software ESPRESSO [35] and is implemented in C. In fact, two variants of the algorithm were used for the benchmarks. In the first benchmarks, the algorithms did not yet employ interlacing, and the $i\boldsymbol{k}$-differentiation scheme was used for computing forces onto particles. During the time of writing the implementation has been extended to use both analytical differentiation and interlacing, as it was determined that this is the fastest combination of components [51]. Since JUGENE was replaced with a new architecture before these changes were implemented, it was not possible to rerun all benchmarks with the modified algorithm. In the graphs, whenever the noninterlaced method with ik differentiation was used, it is denoted as “P3M (ik).” The second implementation of a Fourier-based method is P^2NFFT; see [50]. Starting from the fast summation method [58], it was shown in [61] that the method is based mainly on a “convolution at nonequispaced nodes.” This conclusion leads to great simplicity: The P^2NFFT implementation mainly consists of only two building blocks required to compute this convolution, namely, the FFT [63] and the NFFT [50].

As noted in Sec. II A2, the implementations of P^3M and P^2NFFT are mathematically equivalent for periodic boundary conditions. Whenever there are differences found in the performance between these two, they originate only in part from the implementation itself, but mainly from differences in the way in which the parameters, which enter into the algorithms, were determined. For all timings of P^2NFFT, all parameters were chosen based on the comparison of several runs, whereas for P^3M, an estimate for the near-field cutoff r_{cut} and an automatic tuning of all other parameters is used.

The core of the FMM is implemented in FORTRAN 90. Sorting is done externally via a call to a C library function. The parallel version of the FMM does make use of the message passing interface (MPI) for collectives and ARMCI and OSPRI (Blue Gene/P only) for the one-sided, nonblocking point-to-point communication. To simplify the use of the algorithm for the user, an additional run time and error control scheme was implemented to automatically tune the parameter set.

The implementation of MEMD is written in C and was ported from the Molecular Dynamics software ESPRESSO [35]. It is based on the $\boldsymbol{B}$ field wave propagation version proposed by Dünweg and Pasichnyk [25] rather than the original diffusive Monte Carlo propagation [24]. The interpolation scheme for the electric currents is based on a linear charge interpolation. The simulation box is divided into subdomains, and each subdomain is mapped onto a grid of equally spaced lattice sites in each dimension and across all domains, which facilitates the applicability of a finite differences curl operator ($\nabla\times$). The parallel communication is merely done on next neighbor boundaries. The algorithm includes dynamics (propagation of fields) and, to set the time scale of the system, requires—in contrast to the other algorithms—the specification of a time step to correctly map the motion of charges to the electric current.

IV. STABILITY

The long-term stability of an MD simulation is a good test for the accuracy of the computed forces. If a symplectic integrator is used, the time discretization error does not lead to a long-term energy drift, so that any remaining drift must be due to systematic errors in the forces. A symplectic integrator has the property that the discretized solution for the original Hamiltonian H is equal to the *exact* solution for a nearby Hamiltonian H' [78]. Since the energy for H' is exactly conserved, its deviation from the energy for H remains bounded. Likewise, a time-reversal symmetry of the integrator guarantees the conservation of total momentum. If total momentum is not conserved, there must be systematic errors in the forces.

To evaluate the long-term stability, we have run MD simulations of the silica melt system described in Sec. III A using the BKS force field [72], the Coulomb component of which is computed with the different methods from the SCAFACOS library. The simulations were performed with the MD code IMD [79], using a symplectic leap-frog integrator in the NVE ensemble. The system consisted of 12 960 atoms and was first equilibrated at 2950 °C. The simulations were

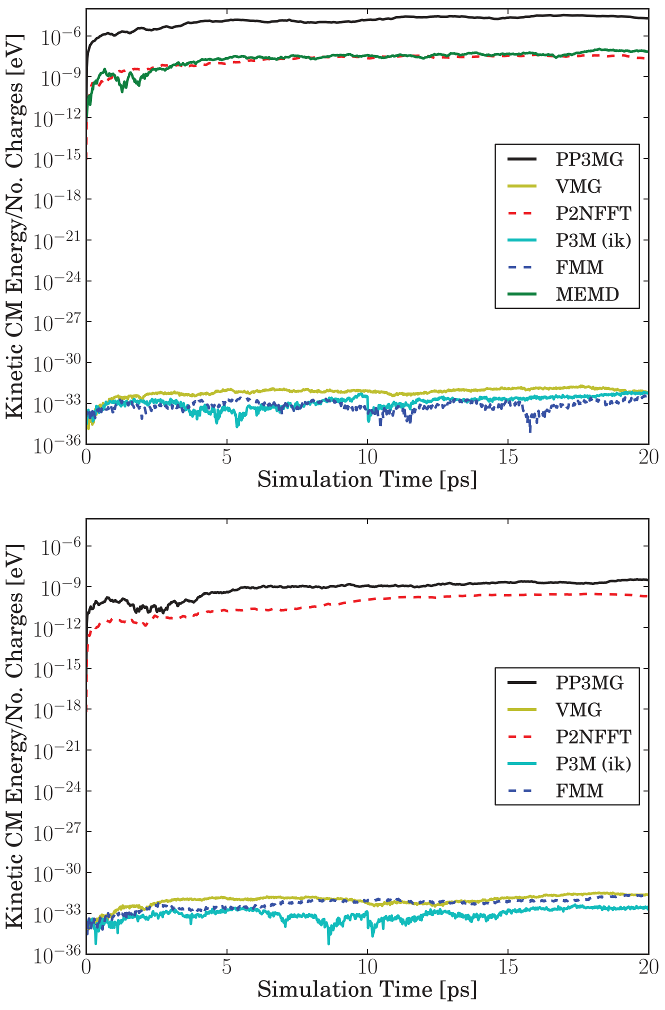

FIG. 6. (Color online) Kinetic energy per particle of the center-of-mass motion of the system according to different methods and accuracies: $\varepsilon_{\text{pot}} = 10^{-3}$ (top); $\varepsilon_{\text{pot}} = 10^{-5}$ (bottom). Note that the P^{3}M method used here uses an ik differentiation for the forces instead of analytical differentiation and does not apply interlacing. With MEMD, only an accuracy of $\varepsilon_{\text{pot}} = 10^{-3}$ could be reached.

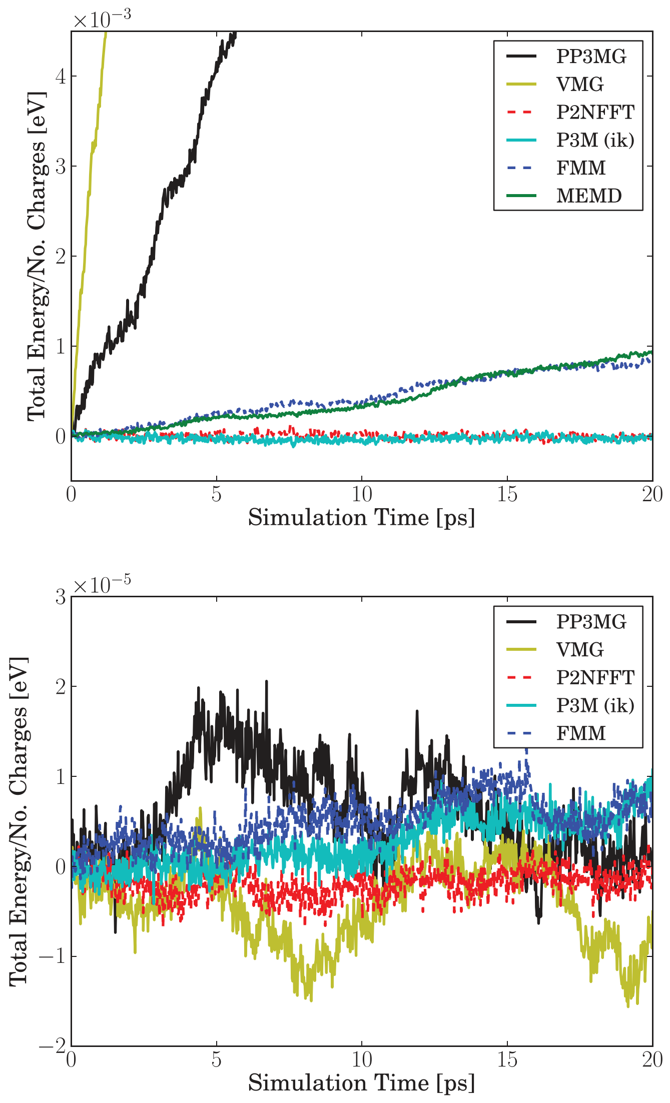

FIG. 7. (Color online) Total energy per particle according to different methods at accuracies $\varepsilon_{\text{pot}} = 10^{-3}$ (top) and $\varepsilon_{\text{pot}} = 10^{-5}$ (bottom). The initial energy has been subtracted in order to make the fluctuations and the drift better visible. Note the different scales of the two figures (units of 10^{-3} and 10^{-5} eV). Note that the P^{3}M method used here uses an ik differentiation for the forces instead of analytical differentiation and does not apply interlacing. With MEMD, only an accuracy of $\varepsilon_{\text{pot}} = 10^{-3}$ could be reached.

run over 100 000 time steps of 0.2 fs each. This time step is a rather small, conservative choice, so that remaining time discretization errors should be negligible.

A. Conservation of momentum

The results for the conservation of momentum are shown in Fig. 6, for two different required accuracies. At the beginning of the simulation, the total momentum was set to zero. The development of the specific kinetic energy of the center-of-mass motion, calculated in energy per particle, is a good measure to monitor the momentum conservation in the system.

As can be seen from Fig. 6, for the methods FMM, P^{3}M, and VMG the center-of-mass momentum remains zero for all practical purposes, whereas with the methods P^{2}NFFT, PP3MG, and MEMD, a small but noticeable increase of the center-of-mass momentum is visible. The reason for the difference between P^{3}M and P^{2}NFFT is the use of different differentiation schemes. The ik-differentiation scheme used in P^{3}M conserves the momentum exactly, but is somewhat slower, whereas the analytical differentiation scheme used in P^{2}NFFT conserves the momentum only approximately but is faster. Both methods can use either of these differentiation schemes. There is a trade-off between higher accuracy and higher performance involved when one selects the differentiation scheme. Note that VMG artificially enforces conservation of momentum while PP3MG does not.

B. Conservation of energy

The conservation of total energy is shown in Fig. 7, for the same required accuracies as for the momentum. For the higher accuracy, the drift of the energy is negligible for all practical purposes, even though the multigrid methods seem to have slightly higher energy fluctuations. The situation is different for the lower required accuracy, where the multigrid methods VMG and PP3MG show a very clear drift. This means that the forces contain a systematic error, which adds up during the simulation. The other methods, especially the Fourier-based ones (P^{3}M and P^{2}NFFT), behave much better in this respect. They show no energy drift (or a much smaller one) even at low accuracy. This must be taken into account in the Methods selection. If a drift must be avoided, the multigrid methods have to be run at higher accuracy, which costs performance, whereas the other methods can be used at lower accuracy without having to deal with a drift.

It is apparent that MEMD is not capable of achieving an accuracy of $\epsilon < 10^{-5}$, which was found to provide long-term stability for the other methods. The lack of accuracy of MEMD is due to a low near-field resolution in directly adjacent cells, where no explicit particle interactions are considered. The application of a thermostat could possibly circumvent the numerical drift at lower accuracy but cannot guarantee the correct description of the dynamics of the system. However, if the main focus is on the configuration of a system and not on dynamics, e.g., in Monte Carlo simulations, MEMD might be considered as a fast method.

V. PERFORMANCE OF THE METHODS

In the following we assess the performance of each method in terms of accuracy, complexity, and parallel scalability. For the accuracy measurement, we inspect how much longer a method runs for a specific increase in desired accuracy of the relative potential error. The complexity tests check on the theoretically expected complexity with respect to the experimentally measured dependency of run time on the number of particles. Finally, parallel scalability extends the benchmarks to very large systems of charges and examines parallel efficiency of the methods up to very large numbers of cores.

A. Accuracy

For the following benchmarks, the parameters of the fast Coulomb solvers were tuned in order to achieve different potential errors ε_{pot}. Figure 8 shows the run time per particle for the cloud-wall system with 102 900 particles on JUROPA. We discuss each method's source of error individually to explain the details of Fig. 8.

The error introduced by the multigrid-based methods (VMG and PP3MG) is composed of a discretization error, an algebraic error, and an interpolation error. The first is due to the discretization of (7) and depends on the chosen discretization scheme. It scales typically like $O(h^2)$, $O(h^4)$, or $O(h^6)$. Further, an appropriate splitting function has to be chosen that, in order to be represented adequately on the grid, has to be smooth enough and must have a large enough support. As a consequence the choice of a higher-order scheme leads to a larger number of grid points a charge is sampled onto. The order of the interpolation scheme is also chosen according to the order of the discretization scheme, this results in a run time behavior similar to the sampling part. The algebraic error due to the iterative solution of the linear system is controlled to be on the same order.

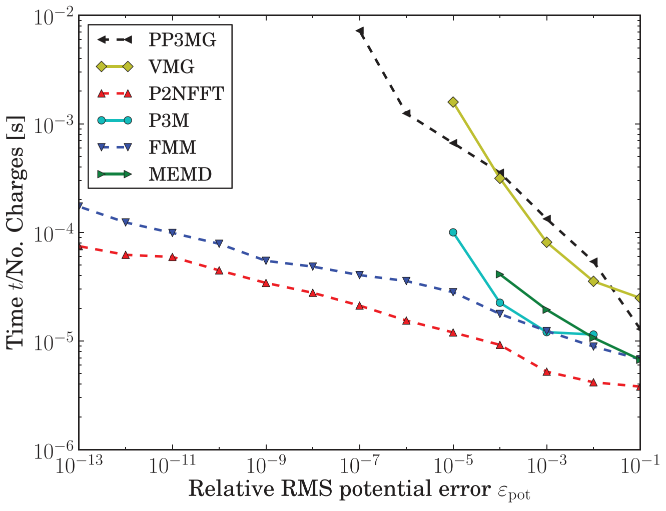

FIG. 8. (Color online) Required wall clock time per particle for 102 900 charges versus the relative rms potential error ε_{pot} on one core of JUROPA.

Putting this information together explains Fig. 8: The grid sizes, inversely proportional to the discretization parameter h, can only be increased by a factor of 2^3 or powers thereof and the resulting error decreases then like the order of the discretization scheme, e.g., $O(h^4)$. However, as the accuracy was measured in powers of 10, not only are the grid sizes changed and the support of the splitting function adjusted accordingly, but also the optimal discretization scheme, interpolation degree, and type of splitting function are chosen individually.

Regarding the Fourier-based methods, the error of the truncated Ewald sum splits into the errors caused by the truncation of the near-field sum (12) at a given near-field cutoff range r_{cut} and the error caused by the truncation of the Fourier series (13) after M mesh points in every direction of space. Note that both errors depend on the Ewald splitting parameter α in the opposite direction; e.g., for every given r_{cut} and M there exists an optimal choice of α for which both errors are balanced. In addition, P^{3}M and P^{2}NFFT spread the charges q_j with a B-spline function on the grid in order to make the problem suitable for FFTs. This introduces another approximation error that decreases exponentially for increasing order P of the B spline. The P^{3}M method seems to be significantly worse than P^{2}NFFT; however, it was noticed that this seems to be caused mostly by problems in the automatic parameter tuning procedure. As the P^{3}M method is equivalent to the P^{2}NFFT method, it can be expected that the method can be improved to perform on the same level as P^{2}NFFT.

The FMM has two distinct error sources contributing to the overall error. Both error sources emerge in the far field only. The FMM near field is free of errors except for numerical rounding off visible in all methods. The first algorithmic error source, the limited number of poles, introduces a truncation error. The second error source occurs whenever the translation operator $\mathcal{O}_{M\to L}$ is applied. Both errors can be controlled in the current implementation such that the results do not show any errors compared to the direct summation or even

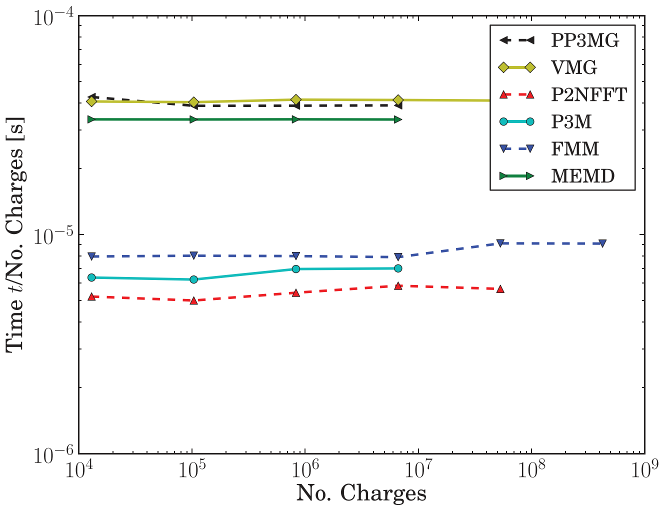

FIG. 9. (Color online) Required wall clock time per particle at relative rms potential error $\varepsilon_{\text{pot}} < 10^{-3}$ on one core of JUROPA.

exceed the precision due to fewer terms in the summation. One cannot state any accuracy scaling properties for systems with arbitrary particle distribution. However, for well behaved, almost homogeneously distributed systems like the ones used in this comparison, the FMM shows the following scaling behavior. First, up until $\Delta E_{\text{rel}} = 10^{-4}$ the run time of the FMM is constant. Second, the run time increases linearly until $\Delta E_{\text{rel}} = 10^{-8}$. Finally, the FMM shows a quadratic scaling behavior until machine precision. The theoretical $O(p^3)$ scaling, with p being the number of poles in the expansion, is not visible when increasing the requested accuracy up to machine precision, since the FMM does tune the near- and far-field contributions automatically which improves the accuracy scaling.

The discretization of the MEMD method introduces two numerical errors, both of which can be expressed using the inverse lattice spacing $1/a$. The first is a charge discretization error that originates in the linear interpolation on next neighbors for the electric current. As expected, it scales with $1/a^3$ and can theoretically be reduced by extending the short-range cutoff of the algorithm to several cells. This, however, implies a high increase in computational effort and does not allow for spatially varying dielectric properties, one of the main advantages of MEMD. The second error is related to the use of retarded solutions of the Maxwell equations with an adjusted speed of light. This in turn can be directly expressed via the lattice spacing, since this dictates the propagation speed of the magnetic fields. From (14) it is apparent that this error scales with a^2. This error formulation yields a minimum error of about 10^{-4} for the relative force error, where realistically 10^{-3} is achievable in most systems. The method provides a tuning function to find the parameters for a minimal error. In Fig. 8, MEMD performs acceptably but can only cope to a maximum precision of 10^{-4} in the relative potential error.

B. Complexity

In this section we compare the theoretical complexity of each method with the measured run time complexity. Figure 9 shows the run time per particle as a function of the number of particles for the various methods on a single core of the JUROPA system with the silica melt test system duplicated as described in Sec. III A. All the methods are required to maintain the relative rms potential error of $\varepsilon_{\text{pot}} < 10^{-3}$. Note that not all the implementations are intended to run such large problem sizes on a single core. Therefore, a lack of memory or other implementation depended limitations lead to some missing data points in Fig. 9.

We observe an almost linear increase of run time with increasing number of particles for all compared methods, which agrees perfectly with the theoretical $O(N)$ complexities of the MEMD, PP3MG, VMG, and FMM methods. The time per particle varies between 5.2×10^{-6} and 1.4×10^{-4}. Although P^{3}M and P^{2}NFFT yield a theoretical $O(N \log N)$ complexity, this is not visible in the compared range of particle numbers. This does not emerge as the run time share of the FFT, which is responsible for the asymptotic $O(N \log N)$ scaling, ranges only between 3% and 10% of the total run time. Further note that there is no crossover point of the FMM and P^{3}M run time up to 5×10^7 particles. However, the run time difference between FMM, P^{3}M, and P^{2}NFFT are rather marginal for all compared system sizes.

C. Scalability

Our last benchmark is focused on parallel scalability. Although each method presented here shows linear scaling behavior for a small number of cores, at some point implementation-dependent restrictions will cause a deviation from the $O(N/P)$ behavior at large problem sizes N and large numbers of cores P and thus a decrease in efficiency for highly parallel execution, as expected for strong scaling. The timings presented in this section were performed on the two different architectures described in Sec. III C. All algorithms are tested for parallel scalability using the cloud-wall test case from Sec. III A on the JUGENE system, which has a large number of slowly clocked cores, and on the JUROPA system, an architecture more similar to common compute clusters. Results of the wall clock measurements for 1 012 500 charges on the JUROPA system are presented in Fig. 10.

The gray dotted lines drawn in Fig. 10 each denote a factor of 10 in the run times with respect to the fastest method on one core. A graph starting out on the lower line therefore is reduced to a relative efficiency of 0.01 when crossing the upper line, providing the best visible resolution in a region of very low efficiency. All methods in Fig. 10 show very similar efficiency and their scaling behavior at low core numbers appears linear.

To provide a better resolved and easier to read comparison, the scaling plots presented from here on will feature the *relative parallel efficiency* in dependence on the number of cores. Let $P_{\min}$ denote the minimal number of cores that was included in the measurements and t_{best} the run time of the fastest method at this level of parallelism. We then plot the *relative parallel efficiency* $e(P)$, which we define by

$$e(P) = \frac{t_{\text{best}}}{t(P)} \frac{P_{\min}}{P}. \quad (15)$$

The same scaling measurements as in Fig. 10 are shown in Fig. 11(c) using Eq. (15). In direct comparison, the nonlinearity

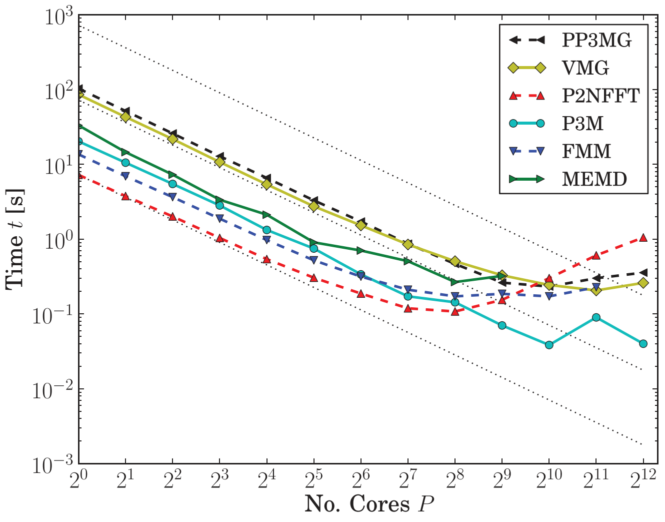

FIG. 10. (Color online) Required wall clock time of the cloud-wall test case with 1 012 500 charges versus the number of cores on JUROPA at relative rms potential error $\varepsilon_{\text{pot}} < 10^{-3}$.

even at small numbers of cores can be seen, and the scaling trend of each algorithm in the most interesting scope above 10% efficiency of the fastest method is more apparent on the nonlogarithmic scale. The only disadvantage of plotting the parallel scaling like this is that it is not possible to read off the actual timing of an algorithm from these plots. To allow for this, we give the best timing t_{best} at the minimal number of cores P_{min} in the caption of the scaling graphs. The actual timing can then be calculated as $t(P) = \frac{t_{\text{best}}}{e(P)} \frac{P_{\text{min}}}{P}$. Where $P_{\text{min}} \neq 1$, we give it explicitly.

We now discuss the scalability of each method individually.

1. *VMG and PP3MG*

On both architectures multigrid methods show a very good scaling behavior. Both methods use a fourth compact order discretization scheme that maximizes parallel efficiency at moderate accuracy. The efficiency plots on both architectures are smooth and the methods behave like expected; cf., Fig. 11 and Fig. 12. On JUROPA (see Fig. 11) the performance of VMG overall is a little bit better than the behavior of PP3MG; nevertheless, both methods show a loss of efficiency for small, i.e., less than 1000, numbers of particles per core. This effect seems to be less pronounced for PP3MG. The factor between VMG and PP3MG can be explained by examining the parameters chosen for VMG, that seem to be chosen more favorable for this implementation. On JUGENE (cf. Fig. 12), both methods perform equally for smaller processor number, for larger processor numbers PP3MG suffers less from scalability issues. Note that both methods were tuned manually.

2. *P^2NFFT and P^3M*

On the JUROPA architecture, P^2NFFT and P^3M outperform the other methods (see Fig. 11). Mathematically, the two methods are identical [65], and for these scalings both use interlacing and the analytic differentiation scheme. Therefore, the significant difference in performance and scaling seen in Fig. 11 stems solely from the different choice in the

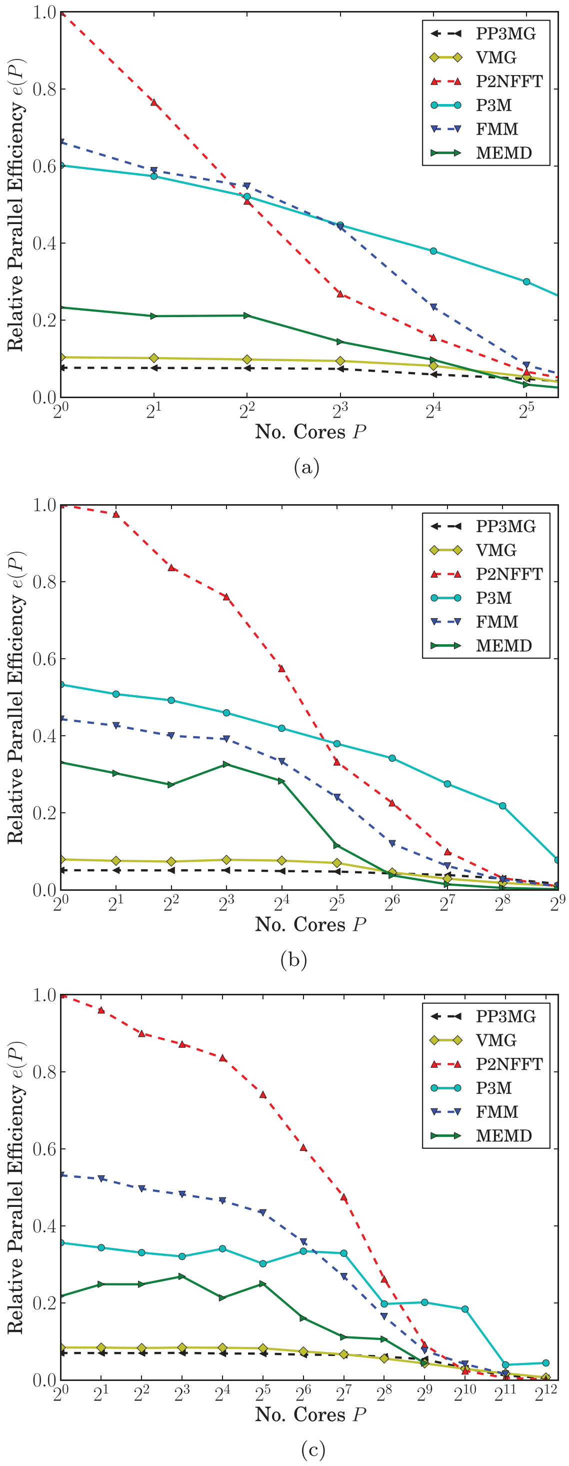

FIG. 11. (Color online) Relative efficiencies $e(P)$ [see (15)] of the cloud-wall test case at different sizes versus the number of cores on JUROPA at relative rms potential error $\varepsilon_{\text{pot}} < 10^{-3}$. (a) Cloud-wall test case with 8100 charges. $t_{\text{best}} = 6.35 \times 10^{-2}$ s. (b) Cloud-wall test case with 102 900 charges. $t_{\text{best}} = 6.60 \times 10^{-1}$ s. (c) Cloud-wall test case with 1 012 500 charges. $t_{\text{best}} = 7.25$ s.

algorithms' specific parameters. In these simulations, P^3M uses an automatic parameter tuning method, whereas P^2NFFT timings were done with manually tuned parameters. The automatic tuning method makes use of the error estimate

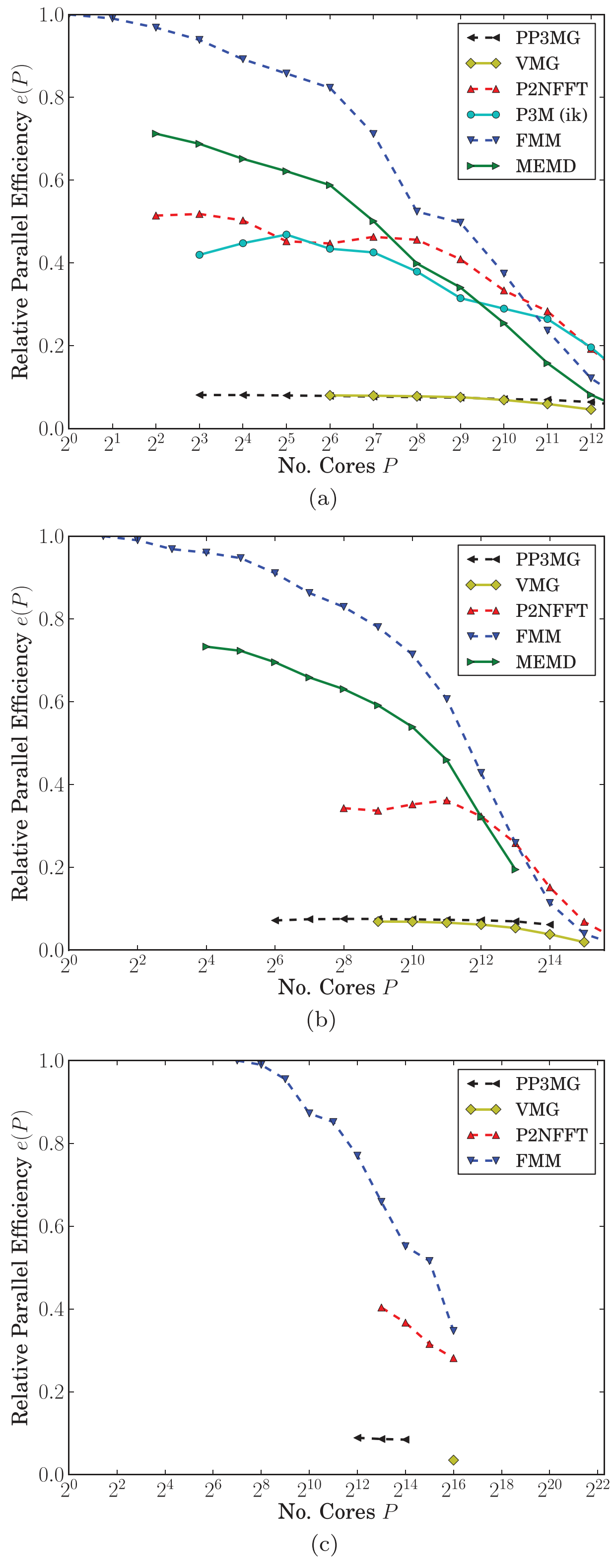

FIG. 12. (Color online) Relative efficiency [see (15)] of the cloud-wall test case with relative rms potential error $\varepsilon_{\text{pot}} < 10^{-3}$ on the JUGENE architecture. Note that the P^{3}M method used here uses an *ik* differentiation for the forces instead of analytical differentiation and does not apply interlacing. (a) Cloud wall test case with 1 012 500 charges. $t_{\text{best}} = 106.7$ s. (b) Cloud wall test case with 9 830 400 charges. $t_{\text{best}} = 454.8$ s; $P_{\min} = 2$. (c) Cloud wall test case with 102 900 000 charges. $t_{\text{best}} = 753.1$ s; $P_{\min} = 16$.

from [52] and times several test runs with different sets of real-space cutoffs and lattice sizes to determine the optimal parameters. This works reasonably well within a full MD software but may give flawed results in a library, where the ratio of computational effort between short-range and long-range interactions is not as obvious.

The nonoptimal tuning routine of P^{3}M results in a considerably larger near-field cutoff range than the manually chosen parameters of P^{2}NFFT, which were optimized for an ideal single core performance. Hence, the P^{2}NFFT calculations are mostly done in Fourier space via the FFTs, which is advantageous on small numbers of cores. On a large number of cores, this situation is reversed and P^{3}M as well as also other methods perform better for the same reason: More Fourier space calculations demand more global communication, whereas in the case of P^{3}M more interactions are calculated in real space with a near-field solver requiring less global communication.

With both implementations, it is possible to switch parameters at the point of crossover, making the resulting algorithm the fastest on the JUROPA architecture.

On the JUGENE architecture (see Fig. 12), both algorithms used an older implementation. Neither of which featured interlacing, and in the case of P^{2}NFFT this included the analytic differentiation scheme, whereas P^{3}M used the ***ik*** differentiation. The automatic tuning routine was used for all timings. P^{2}NFFT, in contrast to scalings on the JUROPA architecture, did not tune the near-field cutoff range. At the time of the scaling measurements, it was not yet possible to run P^{3}M for the setups in plots in Figs. 12(b) and 12(c) because of parallelization issues with uneven automatic domain decomposition of the MPI library. This problem is fixed in the current version of SCAFACOS.

The good scaling behavior (at the expense of slower single core performance because of the large near-field cutoff) observed on the JUROPA architecture is still visible and both implementations surpass the FMM performance for very high numbers of cores.

3. *FMM*

As can be seen in Fig. 12, the FMM implementation performs very well on the JUGENE architecture, showing not only the best single core performance of the compared methods but also excellent scaling behavior, making it the fastest method down to 10^3 particles per core. The OSPRI library is used for point-to-point communication. At large scales, a global communication scheme could still improve the scaling behavior, but point-to-point communication was chosen to optimize the method for a small memory footprint.

On the JUROPA architecture (see Fig. 11), FMM shows good results at small numbers of cores and good scaling behavior, but is outperformed by the Fourier-based methods. Comparison of the two architectures suggests that the ratio of computing time to communication is not optimal since the implementation performs very well on JUGENE with slowly clocked cores. This is partly due to the ARMCI message passing layer not scaling as expected, the communication being one-sided (blocking or nonblocking), and the lack of thread-based parallelism. Overall, it is mostly due to the optimization on small memory footprints.

Of all methods presented here, the implementation of FMM has the lowest memory requirement. Several additional communication steps are required to achieve this. The number of global synchronizations can be reduced if this low memory requirement is lifted. The Morton-ordering and sorting algorithm is restricted to a small memory requirement. This ordering also introduces imbalance; hence the scaling limit. Additional load balancing would be needed to even out the reordering effects.

4. MEMD

The MEMD algorithm performs acceptably on the JUROPA system. The absolute timings do not reach those of the fastest methods but are situated in the midrange compared to all alternatives. The parallel scaling is not as strong as expected from a purely local method, which is due to the large amount of data sent between cells at the node boundaries. In addition to interpolated charges as in other methods, currents and electric and magnetic fields have to be exchanged, which leads to a communication overhead on clusters with good single CPU performance.

On a system with slower clocked cores like JUGENE, MEMD performs quite competitively, as can be seen in Fig. 12. A comparison between the two architectures reveals room for improvement within the communication structures.

The timings shown here are all measurements of the dynamic solution of the algorithm and do not include the first time step, which is calculated via the initial numerical relaxation scheme. Since MEMD does not provide a tuning method for a given error estimate, manually tuned parameters were used for all simulations to obtain the required accuracy at the coarsest possible mesh. On the JUGENE architecture, due to memory consumption, the simulations had to be restricted to higher numbers of cores.

D. Performance comparison

As a general remark we emphasize that the comparison was made between methods included in the parallel library SCAFACOS. Due to the common interfaces, positions, and charges are copied into the library and resorted internally. Although there are various kinds of optimizations already done, a performance gain could be obtained by transferring sorted arrays of charges and positions of particles in, e.g., block structures or along space filling curves to the library for an improved cache usage. Furthermore, the evaluation of the short-range part of interactions within the force loop of the MD code has the potential to increase performance due to the combined calculation with empirical potentials such as Lennard-Jones, which avoids a double calculation of mutual distances between near-field particles in the MD code and in the library. For the present comparison the implementation of the near-field contribution was calculated consistently with the methods offered by SCAFACOS in order to put all methods on a common ground. Finally, changes to particle position in an MD simulation are usually small such that especially the PDE-based methods may benefit from starting at solutions gained from the previous time step. However, such optimizations have not been explored for this comparison for the sake of a common ground.

The systems chosen for the benchmarks fulfill the requirements to exhibit a sufficient contribution from long-range contributions to the electrostatic potential, whereas the distribution of particles is relatively homogeneous. It has to be pointed out that for systems which show strong inhomogeneous particle distributions, mesh-free methods like FMM will most likely gain in relative performance with respect to mesh-based methods, like P^2NFFT and P^3M, since a sufficient resolution of the particle distribution will call for large meshes.

It is apparent that for architectures consisting of powerful single cores, the FFT-based methods P^2NFFT and P^3M show the best performance of all methods included in the SCAFACOS library. It can be seen that the chosen parameters are crucial for accuracy and speed. For example, the real-space cutoff radius might be set small when a small number of processors is used, since the far-field part can then be very efficiently calculated by a well scaling FFT. However, when the number of cores is increased and the efficiency of the FFT is degraded due to communication, more work is transferred to the short-range part by finding a balance between far-field and near-field calculations by a larger near-field cutoff radius.

For architectures with a smaller ratio of communication to computation time, i.e., a fast network structure together with slow single cores, the fastest algorithm is the FMM. It also features the smallest memory requirements, allowing for very large systems even on small numbers of cores. The implementation has been optimized for a small memory footprint, and it is most likely that the timings can even be further improved by use of a less memory efficient sorting and communication schemes. For a small error threshold ($\varepsilon < 10^{-5}$), controlling the approximation, the given implementation of the FMM is sufficiently energy and momentum conserving for long trajectory calculations. A further advantage of the FMM, that was not explored in the present article, is its ability to deal with partially periodic systems.

The multigrid methods perform acceptably and show a very good scaling behavior. The only global communication which is required is the reduction of the defect over all cores, which is necessary for controlling the overall convergence of the method. Further optimization could reduce the workload of local computations, i.e., charge assignment and near-field correction. Although the method shows a moderate performance compared with other methods at the same accuracy, the underlying multigrid solvers are flexible enough to allow for many variations, e.g., spatially varying dielectric properties of the Poisson equation, which makes them attractive for a broader class of applications.

MEMD is a recently developed method that, especially on architectures with a favorable ratio of single core performance and communication performance, can compete with the more established algorithms presented in this article. The performance at higher accuracies remains an open question but is, in principle, achievable. Methodically, MEMD is highly scalable and, with an improved and optimized implementation of the communication structures, could be an interesting option for applications including spatially varying dielectric properties. Nevertheless, MEMD in its current state shows some limits in applicability. First, for algorithmic reasons, it does not perform well for inhomogeneous systems or systems with vast changes between consecutive configurations. Second, for physical reasons it cannot deal with systems that feature spatially fixed particles or external driving fields that would result in a net electric current. In its present implementation and development state it cannot be tuned to very high precision, mainly because of its limited resolution of short-range interactions.

VI. SUMMARY AND CONCLUSIONS

The present article focused on a comparison of algorithms for the calculation of long-range (electrostatic) interactions in many-particle systems with periodic boundary conditions in terms of stability, accuracy, complexity, and parallel scalability.

Stability. As expected for a symplectic integrator, the center-of-mass momentum is conserved for FMM-based as well as FFT-based methods (when applying differentiation in Fourier space) within the applied error bounds. Small deviations from momentum conservation are observed for FFT - and multigrid-based methods, which apply an analytic differentiation scheme (unless the momentum is artificially removed, as is the case for VMG).

The total energy of the system is conserved very well for simulations with a relative rms potential error of $\varepsilon_{\mathrm{pot}} \leqslant 10^{-5}$ for all methods except MEMD, which does not reach the accuracy level. At a lower relative precision of 10^{-3}, the FFT-based methods P^{3}M and P^{2}NFFT conserve the total energy of the system perfectly for methodical reasons. The FMM shows a slight systematic deviation, and both multigrid methods exhibit a clear energy drift that needs to be corrected for.

Complexity. All implementations behave as expected from a theoretical analysis of the algorithms. The $\log(N)$ contribution of the FFT-based methods is not visible for common system sizes, as the work spent in the FFT is negligible compared with other tasks.

Accuracy. The relative increase in run time as a function of reduced approximation error of the methods is smallest for FMM and P^{2}NFFT, which makes them favorable for high-precision calculations. All other methods show a significant increase in computational effort for high-precision force calculations. Differences found here between P^{2}NFFT and P^{3}M can be recast to a different way of determining the parameters, relevant for the accuracy of the methods.

Multigrid methods, which discretize the Laplace operator by high-order finite difference stencils, suffer from a relatively large discretization error compared with Fourier-based methods, which compensate the discretization error by a nonlocal correction via the influence function. Since multigrid methods perform local operations, nonlocal corrections cannot be considered in a natural way.

As mentioned, MEMD is a rather recent method which still awaits maturity. Accuracy of this method at its present stage is moderate and precision requirements of $\varepsilon_{\mathrm{pot}} < 10^{-5}$ could not be reached within the present scope. This can be recast to a rather coarse treatment of short-range interactions, where nearby particles in different cells interact via their discretizations on grid points. In the future this restriction might be lifted to bring also MEMD closer to other methods.

Scalability. A comparison shows very good parallel scaling behavior for both multigrid methods and the automatically tuned P^{3}M. The FMM and MEMD scale very well on architectures with slowly clocked CPUs, but are outperformed on other systems.

On architectures with slowly clocked CPUs, like the JUGENE architecture, the FMM performs fastest among all compared methods. On systems with better single core performance, the FFT-based methods P^{2}NFFT and P^{3}M are the fastest, for a small and large real-space cutoff radius, respectively. The overall best scaling behavior is seen from the multigrid-based methods although they are not the fastest methods.

Although part of the presented algorithms exist in various implementations in the scientific community and are widely used in different simulation packages, a concise comparison has been awaited and results presented here have revealed some common aspects and differences. Though the FMM is currently not widely adopted, it performed very competitively in our implementation. Also the Fourier-based methods scale unexpectedly well to large numbers of charges and cores. In addition, it is apparent that the choice of parameters for each method, while being highly complex, has a significant influence on the accuracy and performance of the algorithms.

All algorithms featured in the present article were compared within the publicly available open source library SCAFACOS [36]. This not only makes it possible to perform a fair comparison between implementations within the same framework but gives the scientific community the possibility to link the library to other software, thereby extending either the functionality or achieving a better scalability. The library is still under development and further optimizations and adaptations to future architectures are planned. Furthermore, it is expected that new approaches to the problem of long-range interactions, exhibiting advantages in various respects (e.g., accuracy, scalability, functionality) can be included into the open source library, which might allow for an even wider comparison in future.

ACKNOWLEDGMENTS

This work was supported by the German Ministry of Science and Education (BMBF) under Grant No. 01IH08001. Additionally, A.A., F.F., and C.H. thank the German Science Foundation (DFG) for support via the SFB 716 program. We are grateful to the Jülich Supercomputing Center for providing the computational resources on Jülich Blue Gene/P (JUGENE) and Jülich Research on Petaflop Architectures (JUROPA). The authors wish to express their gratitude to Lukas Arnold, Paul Gibbon, Alexey Neelov, Pedro Sanchez, Florian Weik, Lidia Westphal, Ralf Wildenhues, and Mathias Winkel for their valuable contributions to the SCAFACOS library.

APPENDIX: SCALING PARAMETERS

1. Scaling, tuning, parameters, compilers, versions

The results presented in this paper were timed with a generic example program that is included within the SCAFACOS library and requires a specific set of parameters per method as command-line input. All simulations were done with the SCAFACOS library version 0.1, published on June 11, 2013, or prior versions (scalings on JUGENE). The method parameters that lead to the timings presented in Sec. III are listed in the following tables. For information on precise names and functions of the set of parameters within each method, we refer to the library manual [76].

Some parameters remain constant through all scaling experiments of one method. For better visibility of the actual changes, these parameters will only be spelled out for the first value and afterward be referred to as "...".

2. JUGENE scaling parameters

The parameters used for the scaling of SCAFACOS on JUGENE with the cloud-wall system were

Method	No. of charges	Parameters
MEMD	1 012 500	mesh = 128, light speed = 0.7, time step = 0.01
	9 830 400	mesh = 256, light speed = 0.25, ...
	102 900 000	mesh = 512, light speed = 0.1, ...
P^2NFFT	1 012 500	$M = 256$, r_cut = 4.481, $P = 4$, $\alpha = 0.573$
	9 830 400	$M = 512$, ...
	102 900 000	$M = 1024$, ...
P^3M	1 012 500	r_cut = 4.481 387, grid = 256, cao = 4, $\alpha = 0.573\,076$
VMG	1 012 500	max_level = 7, near_field_cells = 5, discretization_order = 4, interpolation_order = 5, precision = 1.0e-4, smoothing_steps = 3
	9 830 400	max_level = 8, ...
	102 900 000	max_level = 9, ...
PP3MG	1 012 500	cells_x = cells_y = cells_z = 256, ghosts = 6, degree = 5
	9 830 400	cells_x = cells_y = cells_z = 512, ...
	102 900 000	cells_x = cells_y = cells_z = 1024, ...
FMM	1 012 500	tolerance_energy = 0.001
	9 830 400	tolerance_energy = 0.003
	102 900 000	tolerance_energy = 0.001

3. JUROPA scaling parameters

The parameters used for the scaling of SCAFACOS on JUROPA with the cloud-wall system were

Method	No. of charges	Parameters
MEMD	8100	mesh = 16, light speed = 3.0, time step = 0.01
	102 900	mesh = 32, light speed = 2.0, ...
	1 012 500	mesh = 128, light speed = 0.7, ...
P^2NFFT	8100	$M = 32$, r_cut = 2.3, $\alpha = 0.995$, $P = 4$
	102 900	$M = 64$, r_cut = 3.0, $\alpha = 0.811$, ...
	1 012 500	$M = 128$, r_cut = 3.5, $\alpha = 0.711$, ...
P^3M	8100	grid = 32, cao = 3, $\alpha = 0.489\,078$, r_cut = 4.48139
	102 900	grid = 64, cao = 4, $\alpha = 0.573\,076$, ...
	1 012 500	grid = 256, cao = 4, $\alpha = 0.611\,589$, ...
VMG	8100	max_level = 5, near_field_cells = 5, discretization_order = 4, interpolation_order = 5, precision = 1.0e-4, smoothing_steps = 3
	102 900	max_level = 6, ...
	1 012 500	max_level = 7, ...
PP3MG	8100	cells_x = cells_y = cells_z = 32, ghosts = 5, degree = 5, tol = 1e-4, max_iterations = 50, discretization = 1, distribution = 1
	102 900	cells_x = cells_y = cells_z = 64, ghosts = 6, ...
	1 012 500	cells_x = cells_y = cells_z = 128, ghosts = 5, ...
FMM	8100	tolerance_energy = 0.003
	102 900	tolerance_energy = 0.0005
	1 012 500	tolerance_energy = 0.001

4. Accuracy parameters

The parameters used for the accuracy scaling of SCAFACOS with the cloud-wall system were

Method	Accuracy	Parameters
MEMD	10^{-1}	mesh = 16, light speed = 3.0, time step = 0.01
	10^{-2}	mesh = 24, light speed = 2.0, . . .
	10^{-3}	mesh = 32, light speed = 1.5, . . .
P^2NFFT	10^{-1}	$M = 32$, r_cut = 2.0, $P = 4$, $\alpha = 0.580$
	10^{-2}	$M = 64$, r_cut = 2.0, $P = 4$, $\alpha = 0.812$
	10^{-3}	$M = 64$, r_cut = 3.0, $P = 4$, $\alpha = 0.817$
	10^{-4}	$M = 64$, r_cut = 4.0, $P = 6$, $\alpha = 0.645$
	10^{-5}	$M = 128$, r_cut = 3.1, $P = 6$, $\alpha = 0.942$
	10^{-6}	$M = 128$, r_cut = 3.3, $P = 8$, $\alpha = 1.079$
	10^{-7}	$M = 128$, r_cut = 3.7, $P = 10$, $\alpha = 0.963$
	10^{-8}	$M = 128$, r_cut = 5.3, $P = 10$, $\alpha = 0.738$
	10^{-9}	$M = 128$, r_cut = 6.2, $P = 10$, $\alpha = 0.676$
	10^{-10}	$M = 128$, r_cut = 7.2, $P = 10$, $\alpha = 0.626$
	10^{-11}	$M = 256$, r_cut = 4.0, $P = 10$, $\alpha = 1.166$
	10^{-12}	$M = 256$, r_cut = 4.8, $P = 10$, $\alpha = 1.032$
	10^{-13}	$M = 256$, r_cut = 6.4, $P = 10$, $\alpha = 0.818$
P^3M	10^{-2}	grid = 64, cao = 3, $\alpha = 0.489\,078$, r_cut = 4.481 39
	10^{-3}	grid = 64, cao = 4, $\alpha = 0.573\,076$, . . .
	10^{-4}	grid = 128, cao = 5, $\alpha = 0.639\,184$, . . .
	10^{-5}	grid = 256, cao = 5, $\alpha = 0.746\,81$, . . .
VMG	10^{-1}	near_field_cells = 2, max_level = 5, discretization_order = 2, interpolation_order = 3, precision = 1.0e-1, smoothing_steps = 2
	10^{-2}	near_field_cells = 3, max_level = 6, discretization_order = 4, interpolation_order = 5, precision = 1.0e-4, smoothing_steps = 3
	10^{-3}	near_field_cells = 5, max_level = 6, discretization_order = 4, interpolation_order = 5, precision = 1.0e-4, smoothing_steps = 3
	10^{-4}	near_field_cells = 10, max_level = 7, discretization_order = 4, interpolation_order = 5, precision = 1.0e-4, smoothing_steps = 2
	10^{-5}	near_field_cells = 18, max_level = 8, discretization_order = 4, interpolation_order = 4, precision = 1.0e-5, smoothing_steps = 2
PP3MG	10^{-1}	cells_x = cells_y = cells_z = 64, ghosts = 3, tol = 1e-2, discretization = distribution = 0, degree = 1, max_iterations = 50
	10^{-2}	. . . , ghosts = 5, tol = 1e-3, discretization = distribution = 1, degree = 3, . . .
	10^{-3}	. . . , ghosts = 6, tol = 1e-4, discretization = distribution = 2, degree = 5, . . .
	10^{-4}	cells_x = cells_y = cells_z = 128, ghosts = 9, tol = 1e-5, . . .
	10^{-5}	. . . , ghosts = 14, tol = 1e-6, . . .
	10^{-6}	. . . , ghosts = 19, tol = 1e-7, . . .
	10^{-7}	cells_x = cells_y = cells_z = 256, ghosts = 29, tol = 1e-8, . . .
FMM	10^{-1}	tolerance_energy = 0.05
	10^{-2}	tolerance_energy = 0.005
	10^{-3}	tolerance_energy = 0.0005
	10^{-4}	tolerance_energy = 0.0001
	10^{-5}	tolerance_energy = 0.000 01
	10^{-6}	tolerance_energy = 0.000 001
	10^{-7}	tolerance_energy = 0.000 000 06
	10^{-8}	tolerance_energy = 0.000 000 005
	10^{-9}	tolerance_energy = 0.000 000 001
	10^{-10}	tolerance_energy = 0.000 000 000 02
	10^{-11}	tolerance_energy = 0.000 000 000 002
	10^{-12}	tolerance_energy = 0.000 000 000 000 3
	10^{-13}	tolerance_energy = 0.000 000 000 000 01

5. Complexity scaling parameters

The parameters used for the complexity scaling of SCAFACOS with the silica melt system were

Method	No. of charges	Parameters
MEMD	12 960	mesh = 24, light speed = 2.0, time step = 0.01
	103 680	mesh = 48, light speed = 1.0, . . .
	829 440	mesh = 96, light speed = 0.5, . . .
	6 635 520	mesh = 192, light speed = 0.25, . . .
P^2NFFT	12 960	M = 32, r_cut = 4.8, P = 4, α = 0.47
	103 680	M = 64, . . .
	829 440	M = 128, . . .
	6 635 520	M = 256, . . .
	53 084 160	M = 512, . . .
P^3M	12 960	grid = 32, r_cut = 5.400 000, cao = 4, α = 0.418 014
	103 680	grid = 64, . . .
	829 440	grid = 128, . . .
	6 635 520	grid = 256, . . .
VMG	12 960	near_field_cells = 4, max_level = 5, discretization_order = 4, interpolation_order = 4, precision = 1.0e-4, smoothing_steps = 2
	103 680	. . . , max_level = 6, . . .
	829 440	. . . , max_level = 7, . . .
	6 635 520	. . . , max_level = 8, . . . , interpolation_order = 3, . . .
	53 084 160	. . . , max_level = 9, . . .
	424 673 280	. . . , max_level = 10, . . .
PP3MG	12 960	cells_x = cells_y = cells_z = 32, ghosts = 4, degree = 3, tol = 1e-4, max_iterations = 50, discretization = 1, distribution = 1
	103 680	cells_x = cells_y = cells_z = 64, . . .
	829 440	cells_x = cells_y = cells_z = 128, . . .
	6 635 520	cells_x = cells_y = cells_z = 256, . . .
FMM	12 960	tolerance_energy = 0.008
	103 680	. . .
	829 440	. . .
	6 635 520	. . .
	53 084 160	. . .
	424 673 280	. . .

[1] P. H. Hünenberger and J. A. McCammon, J. Chem. Phys. **110**, 1856 (1999).
[2] P. P. Ewald, Ann. Phys. **369**, 253 (1921).
[3] D. Fincham, Mol. Simul. **13**, 1 (1994).
[4] J. W. Perram, H. G. Petersen, and S. W. de Leeuw, Mol. Phys. **65**, 875 (1988).
[5] J. Lekner, Physica A **176**, 485 (1991).
[6] R. Sperb, Mol. Simul. **13**, 189 (1994).
[7] F. Dehez, M. Martins-Costa, D. Rinaldi, and C. Millot, J. Chem. Phys. **122**, 234503 (2005).
[8] A. J. C. Ladd, Mol. Phys. **36**, 463 (1978).
[9] R. Sperb, Mol. Simul. **20**, 179 (1998).
[10] R. Sperb, Mol. Simul. **22**, 199 (1999).
[11] A. Arnold and C. Holm, Comput. Phys. Commun. **148**, 327 (2002).
[12] A. Arnold and C. Holm, J. Chem. Phys. **123**, 144103 (2005).
[13] A. Grzybowski, E. Gwóźdź, and A. Bródka, Phys. Rev. B **61**, 6706 (2000).
[14] M. Porto, J. Phys. A: Math. Gen. **33**, 6211 (2000).
[15] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (Taylor & Francis, Bristol, PA, 1988).
[16] T. Darden, D. York, and L. Pedersen, J. Chem. Phys. **98**, 10089 (1993).
[17] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, J. Chem. Phys. **103**, 8577 (1995).
[18] M. Deserno and C. Holm, J. Chem. Phys. **109**, 7678 (1998).
[19] V. Ballenegger, J. J. Cerdà, and C. Holm, J. Chem. Theory Comput. **8**, 936 (2012).
[20] M. Deserno and C. Holm, J. Chem. Phys. **109**, 7694 (1998).
[21] J. E. Barnes and P. Hut, Nature (London) **324**, 446 (1986).
[22] L. Greengard and V. Rokhlin, J. Comput. Phys. **73**, 325 (1987).
[23] K. N. Kudin and G. E. Scuseria, J. Chem. Phys. **121**, 2886 (2004).
[24] A. C. Maggs and V. Rossetto, Phys. Rev. Lett. **88**, 196402 (2002).
[25] I. Pasichnyk and B. Dünweg, J. Phys.: Condens. Matter **16**, 3999 (2004).

[26] A. Arnold and C. Holm, in *Advanced Computer Simulation Approaches for Soft Matter Sciences*, edited by C. Holm and K. Kremer, Advances in Polymer Sciences Vol. II (Springer, Berlin, 2005), pp. 59–109.
[27] G. A. Cisneros, V. Babin, and C. Sagui, *Biomolecular Simulations* (Springer, Berlin, 2013), pp. 243–270.
[28] D. Frenkel and B. Smit, *Understanding Molecular Simulation*, 2nd ed. (Academic Press, San Diego, 2002).
[29] C. Sagui and T. A. Darden, in *Simulation and Theory of Electrostatic Interactions in Solution*, edited by L. R. Pratt and G. Hummer (AIP, Melville, NY, 1999), pp. 104–113.
[30] G. Sutmann, P. Gibbon, and T. Lippert (eds.), *Fast Methods for Long Range Interactions in Complex Systems* (IAS, Jülich, 2009).
[31] A. Y. Toukmaji and J. A. Board, Jr., Comput. Phys. Commun. **95**, 73 (1996).
[32] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, J. Chem. Theory Comput. **4**, 435 (2008).
[33] S. J. Plimpton, J. Comput. Phys. **117**, 1 (1995).
[34] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, J. Comput. Chem. **26**, 1781 (2005).
[35] ESPRESSO—Extensible Simulation Package for Research on Soft Matter, http://espressomd.org
[36] SCAFACOS—Scalable Fast Coloumb Solvers, http://www.scafacos.de
[37] B. Luty, M. Davis, I. Tironi, and W. van Gunsteren, Mol. Simul. **14**, 11 (1994).
[38] A. Y. Toukmaji and J. A. Board, Jr., Comput. Phys. Commun. **95**, 73 (1996).
[39] E. L. Pollock and J. Glosli, Comput. Phys. Commun. **95**, 93 (1996).
[40] K. Esselink, Comput. Phys. Commun. **87**, 375 (1995).
[41] S. W. de Leeuw, J. W. Perram, and E. R. Smith, Proc. R. Soc. London Ser. A **373**, 57 (1980).
[42] E. R. Smith, Proc. R. Soc. London, Ser. A **375**, 475 (1981).
[43] M. Neumann, Mol. Phys. **50**, 841 (1983).
[44] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical Simulation in Molecular Dynamics—Numerics, Algorithms, Parallelization, Applications*, 1st ed. (Springer-Verlag, Heidelberg, 2007).
[45] W. Spotz and G. Carey, Numer. Methods Partial Differ. Equations **12**, 235 (1996).
[46] G. Sutmann and B. Steffen, J. Comput. Appl. Math. **187**, 142 (2006).
[47] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid* (Academic Press, San Diego, 2000).
[48] S. W. de Leeuw, J. W. Perram, and E. R. Smith, Proc. R. Soc. London Ser. A **373**, 27 (1980).
[49] J. W. Perram, H. G. Petersen, and S. W. De Leeuw, Mol. Phys. **65**, 875 (1988).
[50] M. Pippig and D. Potts, SIAM J. Sci. Comput. **35**, C411 (2013).
[51] F. Weik, Master's thesis, Universität Stuttgart, 2011.
[52] V. Ballenegger, J. J. Cerdà, O. Lenz, and C. Holm, J. Chem. Phys. **128**, 034109 (2008).
[53] J. J. Cerdà, V. Ballenegger, O. Lenz, and C. Holm, J. Chem. Phys. **129**, 234104 (2008).
[54] A. Neelov and C. Holm, J. Chem. Phys. **132**, 234103 (2010).
[55] H. Q. Ding, R. D. Ferraro, and D. B. Gennery, *A Portable 3D FFT Package for Distributed-Memory Parallel Architectures* (SIAM, Philadelphia, 1995), pp. 70–71.
[56] J. Keiner, S. Kunis, and D. Potts, ACM Trans. Math. Software **36**, 1 (2009).
[57] D. Potts, G. Steidl, and M. Tasche, in *Modern Sampling Theory: Mathematics and Applications*, edited by J. J. Benedetto and P. J. S. G. Ferreira (Birkhäuser, Boston, 2001), pp. 247–270.
[58] D. Potts and G. Steidl, SIAM J. Sci. Comput. **24**, 2013 (2003).
[59] D. Potts, G. Steidl, and A. Nieslony, Numer. Math. **98**, 329 (2004).
[60] F. Hedman and A. Laaksonen, Chem. Phys. Lett. **425**, 142 (2006).
[61] M. Pippig and D. Potts, in *Fast Methods for Long-Range Interactions in Complex Systems*, edited by G. Sutmann, P. Gibbon, and T. Lippert, IAS-Series (Forschungszentrum, Jülich, 2011), pp. 131–158.
[62] M. Pippig, PNFFT, PNFFT, Parallel Nonequispaced FFT subroutine library, http://www.tu-chemnitz.de/∼mpip/software.
[63] M. Pippig, SIAM J. Sci. Comput. **35**, C213 (2013).
[64] M. Pippig, PFFT, Parallel FFT subroutine library, http://www.tu-chemnitz.de/∼mpip/software.
[65] A. Arnold and M. Pippig (unpublished).
[66] L. Greengard and V. Rokhlin, Acta Numer. **6**, 229 (1997).
[67] C. A. White and M. Head-Gordon, J. Chem. Phys. **101**, 6593 (1994).
[68] H. Dachsel, M. Hofmann, and G. Rünger, *Proceedings of the 13th International Euro-Par Conference, 4641 LNCS* (Springer, Berlin, 2007), pp. 695–704.
[69] C. A. White and M. Head-Gordon, J. Chem. Phys. **105**, 5061 (1996).
[70] H. Dachsel, J. Chem. Phys. **132**, 119901 (2010).
[71] F. Fahrenberger and C. Holm, arXiv:1309.7859.
[72] B. W. H. van Beest, G. J. Kramer, and R. A. van Santen, Phys. Rev. Lett. **64**, 1955 (1990).
[73] JUQUEEN web site, http://www.fz-juelich.de/ias/jsc/juqueen
[74] JUGENE—Jülich Blue Gene/P, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUGENE/JUGENE_node.html
[75] JUROPA—Jülich Research on Petaflop Architectures, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUROPA/JUROPA_node.html
[76] M. Bolten, F. Fahrenberger, F. Gähler, R. Halver, F. Heber, M. Hofmann, I. Kabadshow, O. Lenz, and M. Pippig, SCAFACOS *Manual*, http://www.scafacos.de.
[77] M. Bolten, *Multigrid Methods for Structured Grids and Their Application in Particle Simulation* (Bergische Universität Wuppertal, Wuppertal, 2008).
[78] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations* (Springer, Heidelberg, 2002).
[79] J. Stadler, R. Mikulla, and H. R. Trebin, Int. J. Mod. Phys. C **8**, 1131 (1997).