A new algorithm for electrostatic interactions in Monte Carlo simulations of charged particles

William Robert Saunders[a], James Grant[b], Eike Hermann Müller[c,1,*]

University of Bath, Bath BA2 7AY, Bath, United Kingdom

[a] *Department of Physics*
[b] *Computing Services*
[c] *Department of Mathematical Sciences*

Abstract

To minimise systematic errors in Monte Carlo simulations of charged particles, long range electrostatic interactions have to be calculated accurately and efficiently. Standard approaches, such as Ewald summation or the naive application of the classical Fast Multipole Method, result in a cost per Metropolis-Hastings step which grows in proportion to some positive power of the number of particles N in the system. This prohibitively large cost prevents accurate simulations of systems with a sizeable number of particles. Currently, large systems are often simulated by truncating the Coulomb potential which introduces uncontrollable systematic errors. In this paper we present a new multilevel method which reduces the computational complexity to $\mathcal{O}(\log(N))$ per Metropolis-Hastings step, while maintaining errors which are comparable to direct Ewald summation. We show that compared to related previous work, our approach reduces the overall cost by better balancing time spent in the proposal- and acceptance- stages of each Metropolis-Hastings step. By simulating large systems with up to $N = 10^5$ particles we demonstrate that our implementation is competitive with state-of-the-art MC packages and allows the simulation of very large systems of charged particles with accurate electrostatics.

Keywords: Monte Carlo, electrostatics, particle simulations, computational complexity, Fast Multipole Method

1. Introduction

The accurate representation of all pairwise interactions in classical atomistic simulations is important to minimise systematic errors. In this paper we focus on Monte Carlo (MC) simulations of charged particles. Short-range interactions such as the Lennard-Jones potential can be safely truncated at a finite cutoff distance: when calculating energy differences in a proposed MC move only interactions with a fixed number of other particles in close proximity of the moving particle need to be taken into account. For fixed density the cost of one local Metropolis-Hastings (MH) step is constant, independent of the total number N of particles in the system. However, due to the long-range nature of the Coulomb potential, which decays in proportion to the inverse distance between two charges, including electrostatics is far from trivial since interactions with *all* other particles in the system have to be considered. Worse, interactions with periodic copies or mirror charges have to be taken into account if non-trivial boundary conditions are used.

As the review in [1] shows, a plethora of methods have been developed to address this issue, but care has to be taken to obtain reliable results. In this context the authors of [2] for example find that truncating the Coulomb potential in the MC simulation of water in a highly anisotropic geometry leads to significant systematic errors. Other methods which have been proposed to avoid this problem include solving the

*Corresponding author
[1] email: e.mueller@bath.ac.uk

Poisson equation with a grid-based multigrid method [3], Ewald summation [4] or the naive application of the Fast Multipole Method (FMM) [5, 6, 7]. Unfortunately all those approaches result in a prohibitively large computational cost as the number of particles N in the system grows, typically the cost per MH step increases as $\mathcal{O}(N)$ or $\mathcal{O}(\sqrt{N})$. This renders the simulation of very large systems impossible and limits the predictive power of computer experiments.

This is particularly keenly felt in MC simulations, where in contrast to molecular dynamics, systems evolve through discontinuous moves of individual atoms or small numbers of particles. The current inability to treat electrostatics accurately and efficiently prohibits the application of MC to problems with larger particle counts. On the other hand, in many circumstances it is desirable to simulate a system with MC instead of molecular dynamics. In particular, the grand canonical or semi-grand ensembles where the number of particles can fluctuate are only accessible with MC [8, 9]. Exchanging particle position in so-called 'swap moves' can be far more efficient at equilibrating systems e.g. when simulating the distribution of impurities near grain boundaries. The lack of efficient electrostatic algorithms also limits the use of methods such as phase- and lattice switch MC [10, 11] which allow the accurate determination of free energy differences between phases. Enabling the routine application of MC to larger systems of charged particles will allow researchers to study the physics of a particular system with the computationally most appropriate evolution algorithm.

In this paper we show how the limitations of the MC method for charged particle systems can be overcome by constructing an algorithm which reduces the cost per MH step to $\mathcal{O}(\log(N))$ without sacrificing accuracy, thereby making much larger simulations with realistic electrostatics feasible. Our multilevel approach is inspired by the Fast Multipole method and similar to the method recently proposed in [12]. However, compared to [12] it leads to an overall reduction of computational cost by balancing the time spent in proposing and accepting MC moves in realistic MC simulations.

The key observation motivating our new method is the following: standard FMM constructs local expansions for evaluating the long range interactions on the finest level of the hierarchical tree. While the evaluation of those expansions (and direct interactions with all close by neighbours) in the proposal stage of a MH step is $\mathcal{O}(1)$, re-calculating the local expansions incurs a cost of $\mathcal{O}(N)$ since all local expansions are re-calculated in the upward and downward pass of the algorithm, resulting in an overall $\mathcal{O}(N)$ cost per MH step. By storing the local expansions on each level of the multilevel hierarchy instead of accumulating them on the finest level, the relative cost of the proposal- and accept- stage can be balanced since each of those two steps requires a fixed number of operations on each level of the multilevel hierarchy. As the number of levels L is proportional to $\log(N)$, this results in a total computational complexity per MH step which grows logarithmically in the number of particles.

In summary, the new contributions of our work are as follows:

1. We describe a new hierarchical algorithm for Monte Carlo simulations with accurate electrostatics, which has an $\mathcal{O}(\log(N))$ computational complexity per Metropolis-Hastings step.

2. By comparing to the similar method in [12] we show that our algorithm leads to an overall reduction in cost for realistic Monte Carlo simulations.

3. We describe the efficient implementation of our algorithm in the performance-portable PPMD framework [13] recently developed in our group. Since it has a user-friendly high-level Python interface, PPMD allows the easy implementation of Monte Carlo algorithms with accurate electrostatics.

4. We demonstrate that the runtime of our implementation is competitive with the state-of-the-art MC code DL_MONTE [14, 15], which struggles to simulate systems of the size that are easily accessible with our implementation.

For systems with $N = 10^5$ particles our implementation is an order of magnitude faster than the DL_MONTE code. At this system size, we observe that the alternative approach in [12] results in a 30% longer simulation time than our method. By fitting a semi-empirical model to predict the cost of a simulation as a function of the problem size, we show that asymptotically (i.e. for $N \to \infty$) we expect our algorithm to be twice as fast as the one in [12].

Structure. This paper is organised as follows: After discussing related work in Section 2, we review the native FMM algorithm and describe our new method in Section 3, where we also compare it to the approach in [12]. Following a description of the Python interface for our implementation of the algorithms introduced in this paper in Section 4, numerical results are presented in Section 5. We conclude and outline directions for further work in Section 6.

2. Related work

Methods for including untruncated electrostatic interactions in MC simulations with a computational complexity of $\mathcal{O}(\log(N))$ per MH step have been developed previously in [12, 16]. Compared to our approach, the method in [12] does not construct local expansions, thereby avoiding their recalculation whenever a particle is moved. While this might look like a reasonable simplification, it actually makes evaluating the change in potential for each proposed (but potentially rejected) MC transition more expensive. Since there is typically more than one proposed move per accepted transition, this renders the method in [12] more expensive overall, as our numerical experiments confirm.

A modification of the Barnes-Hut octree algorithm is discussed in [16]. Similar to FMM, the classical Barnes-Hut method constructs a hierarchical mesh structure, and represents the distribution of particles in cells on each level by their multidimensional Taylor expansion coefficients. While the calculation of the total electrostatic energy with the octree algorithm is $\mathcal{O}(N\log(N))$, the authors of [16] present a modification of the method which has a cost of $\mathcal{O}(\log(N))$ per local MC step.

The $\mathcal{O}(\log(N))$ algorithms presented here and in [12, 16] improve on what can be achieved with Ewald summation [4, 17], for which the change in electrostatic energy per MC proposal can be calculated at a computational complexity of $\mathcal{O}(\sqrt{N})$. This is because the overall $\mathcal{O}(N^{3/2})$ cost of the Ewald-based energy calculation is made up by an iteration over all N particles and a sum over $O(\sqrt{N})$ reciprocal vectors (long-range contribution) and neighbouring particles (short-range contribution). If only $\mathcal{O}(1)$ particles move in each proposed move, only a small number of the $\mathcal{O}(\sqrt{N})$ sums have to be evaluated. A similar approach is currently explored in the DL_MONTE code [14, 15], though the implementation at present is limited by the fact that the short-range cutoff of the Ewald summation has to be identical to the cutoff of all other local interactions. In this paper we present numerical results which show that our new method can be used to simulate systems with up to 10^5 charges and accurate electrostatic interactions at a cost of around 1ms per MH step.

For completeness, it should be pointed out that other methods with a computational complexity of $\mathcal{O}(N)$ per MH step have also been developed. For example, in [3] a multigrid method is used to solve the Poisson equation to obtain the electrostatic potential generated by the particles. Since the global electrostatic field has to be re-computed for each (local) particle move and the cost of multigrid grows in proportion to the number of grid points [18], which in turn has to increase with the problem size to ensure an accurate representation of the charge distribution on the computational mesh, the cost is inherently $\mathcal{O}(N)$. An additional disadvantage of the method is that the mapping of the highly peaked charge distribution to a grid introduces interpolation errors.

While all methods discussed so far focus on the development of fast methods for computing the electrostatic interactions between particles based on Gauss' law and the fact that the electric field is the gradient of a scalar potential, a radically different approach is pursued in [19]. The authors of this paper argue that (statistically) integrating over a ficticious traverse electric field in addition to the different particle configurations produces the same results as a traditional Monte Carlo simulation with standard Coulombic interactions. While this enlarges the configuration space, it crucially only requires local updates for each particle move. Since the traverse electric field relaxes rapidly, this only introduces a small overhead, independent of the problem size. Overall the cost of updating all particle positions and the traverse field grows in proportion to N, which results in a $\mathcal{O}(1)$ cost per individual particle move. A non-trivial interpolation scheme is used in [19] to reduce lattice artifacts that are introduced by enforcing Gauss' law on a Cartesian grid.

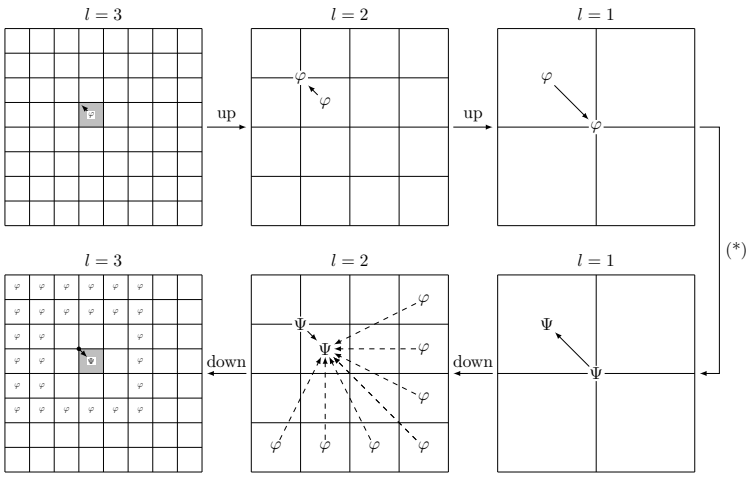

Figure 1: Schematic illustration of the classical Fast Multipole Method in two dimensions first described in [5, 6]. The number of levels is $L = 4$ in this example. The upward pass for constructing the multipole expansions (φ) is shown in the top row, while the local expansions (Ψ) are built in the downward pass at the bottom of the figure. The asterisk (*) on the right hand side denotes special operations on the coarsest level for incorporating (potentially nontrivial) boundary conditions.

3. Method

We now discuss the new approach introduced in this work. After briefly reviewing key concepts of the classical FMM algorithm we describe our method in detail in Section 3.2 and compare it to the related technique in [12] in Section 3.3.

3.1. Fast Multipole method

In three dimensions the FMM algorithm [5, 6, 7] uses a hierarchical grid with L levels for the computational domain Ω (which is assumed to be a cube of width a) such that the number of cells on each level ℓ is $M_\ell = 8^{\ell-1}$ for $\ell = 1, \ldots, L$. The number of cells on the finest level is $M = M_L$, and (to balance the work between the long range and direct field calculation) typically L is chosen such that there are $\mathcal{O}(1)$ particles in each fine level cell. Each cell on level $\ell = 1, \ldots, L-1$ is subdivided into 8 child-cells on the next-finer level; conversely each cell on level $\ell = 2, \ldots, L$ has a unique parent cell. The Fast Multipole Algorithm now computes the electrostatic potential by splitting it into two contributions. First, the long range part is calculated by working out the multipole expansion of all charges in a fine level cell and transforming them into multipole expansions on the coarser levels in the upward pass of the algorithm. In the downward pass the multipole expansions on each level are transformed into local expansions around the centre of a cell. Those are then recursively combined into local expansions in the child cells. By only considering the contribution from multipole expansions in a fixed number of well-separated cells on each level, the potential of distant charges is resolved at the appropriate level of accuracy, while including the contribution from closer charges on finer levels. The method for calculating the long range contribution is shown schematically in Figure 1; we refer the reader to [5, 6, 7] for further details. For the following discussion of our FMM variant for MC simulations the notion of an *interaction list* (IL) of a particular cell is crucial. For a cell α on level ℓ this interaction list $\mathrm{IL}(\alpha)$ is the set of cells which are the children of the parent cell of α and its nearest neighbours, but which are well separated from α, i.e. not direct neighbours of α on level ℓ. Explicitly, the interaction list is defined as

$$\mathrm{IL}(\alpha) = \mathsf{children}\left(\mathcal{N}_b\left(\mathsf{parent}(\alpha)\right)\right) \setminus \left(\alpha \cup \mathcal{N}_b(\alpha)\right),$$

where $\mathcal{N}_b(\alpha)$ is the set of the 26 nearest neighbours of the cell α; the functions $\mathsf{children}(\alpha)$, $\mathsf{parent}(\alpha)$ return the set of child cells or the parent cell of a particular cell α. An example of an interaction list can be found

in the bottom left corner of Figure 1: all cells labelled with the letter φ are in the interaction list of the gray cell labelled with a Ψ.

Finally, interactions with charges in neighbouring fine level cells are included by directly evaluating the $1/r$ potential generated by those charges.

3.2. FMM for Monte Carlo simulations

Now consider the following modification of FMM. Let $\Psi^{\Delta}_{\ell,\alpha}$ be the p-term local expansion around the center of cell α on level ℓ such that $\Psi^{\Delta}_{\ell,\alpha}$ contains contributions from all charges in the interaction list $\mathrm{IL}(\alpha)$ of α. Note that this is different from standard FMM, where the local expansions $\Psi_{\ell,\alpha}$ contain the contribution of all charges not contained in the cell α or its 26 nearest neighbours. However, $\Psi_{\ell,\alpha}$ can be obtained by summing the $\Psi^{\Delta}_{\ell,\alpha}$ on the current and coarser levels, namely

$$\Psi_{\ell,\alpha} = \sum_{\ell'=1}^{\ell} \Psi^{\Delta}_{\ell',\alpha_{\ell'}} \qquad \text{with } \alpha_\ell = \alpha \text{ and } \alpha_{\ell'} = \mathsf{parent}(\alpha_{\ell'+1}) \text{ for all } \ell' = 1,\dots,\ell-1. \tag{1}$$

For a cell α on level ℓ the local expansion $\Psi^{\Delta}_{\ell,\alpha}$ can be expressed in terms of the coefficients $(L^{\Delta}_{\ell,\alpha})^m_n$ as

$$\Psi^{\Delta}_{\ell,\alpha}(\boldsymbol{\delta r}) = \sum_{n=0}^{p} \sum_{m=-n}^{+n} \left(L^{\Delta}_{\ell,\alpha}\right)^m_n (\delta r)^n Y^m_n(\delta\theta,\delta\phi) \qquad \text{with } (\delta r,\delta\theta,\delta\phi) = \mathsf{spherical}(\boldsymbol{\delta r}). \tag{2}$$

Here $\boldsymbol{\delta r}$ is the position of the particle measured relative to the centre $\boldsymbol{R}_\alpha$ of the cell α. The function $\mathsf{spherical}(\boldsymbol{r})$ returns the spherical coordinates (r,θ,ϕ) of a vector $\boldsymbol{r}$. We further call the set $\mathsf{ancestors}(\alpha) = \{\alpha_{\ell-1},\alpha_{\ell-2},\dots,\alpha_2,\alpha_1\}$ defined recursively in Equation (1) the *ancestors* of cell α. Our strategy for evaluating the long range contributions in Monte Carlo simulations is as follows (see Figure 2):

Initialisation. At the beginning of the simulation, calculate the local expansion coefficients $(L^{\Delta}_{\ell,\alpha})^m_n$ for all cells α and on all levels ℓ by using a slightly modified variant of the upward/downward pass in the Fast Multipole Algorithm.

Proposal. Consider a proposed MC move $\boldsymbol{r} \to \boldsymbol{r}'$ of charge q such that the original position $\boldsymbol{r}$ is contained in the fine-level cell α and the new position $\boldsymbol{r}'$ in the fine-level cell α' (which could be identical to α). To evaluate the change in the long range potential, evaluate and accumulate the $\Psi^{\Delta}_{\ell,\alpha_\ell}(\boldsymbol{r}-\boldsymbol{R}_{\alpha_\ell})$ and $\Psi^{\Delta}_{\ell,\alpha'_\ell}(\boldsymbol{r}'-\boldsymbol{R}_{\alpha'_\ell})$ on all levels $\ell = 1,\dots,L$ of the hierarchy by using the sum in Equation (1), where $\boldsymbol{R}_\alpha$ is the centre of cell α. This gives the change in long-range electrostatic energy $\Delta U_{\mathrm{lr}} = q(\Psi_{L,\alpha'}(\boldsymbol{r}'-\boldsymbol{R}_{\alpha'}) - \Psi_{L,\alpha}(\boldsymbol{r}-\boldsymbol{R}_\alpha))$. Secondly, add the change in short-range energy $\Delta U_{\mathrm{direct}}$ from a direct calculation of the interactions with particles in the same and adjacent cells. Finally, remove the spurious self-interaction contribution $q^2/|\boldsymbol{r}'-\boldsymbol{r}|$ which is due to the potential field induced by the charge at the original position. This self-interaction correction is described in detail in [20, Section 3].

Accept. Assume we accept a move $\boldsymbol{r} \to \boldsymbol{r}'$ of charge q such that $\boldsymbol{r}$ lies in cell α and $\boldsymbol{r}'$ in cell α'. For all cells β in the interaction list of α and all its ancestors the contribution of a monopole with charge q is subtracted from $\Psi^{\Delta}_{\ell,\beta_\ell}$ on all levels $\ell = 1,\dots,L$. This requires updating the expansion coefficients $L^{\Delta}_{\ell,\alpha_\ell}$. Conversely, a monopole of charge q is added to the local expansions $\Psi^{\Delta}_{\ell,\beta'_\ell}$ of for all cells β'_ℓ in the interaction list of α' and its ancestors.

The *propose* and *accept* steps are written down explicitly in Algorithms L.1 and L.2; the direct calculation of the interaction with particles in the same fine-level cell or directly adjacent cells to obtain $\Delta U_{\mathrm{direct}}$ is given in Algorithm 4. In Algorithm 4 we remove the spurious self-interaction that occurs between the charge at the proposed position with itself at the original position.

Since the local expansions with $\mathcal{O}(p^2)$ terms need to be evaluated in two cells per level, the cost of one proposal is $\mathcal{O}(p^2 L) = \mathcal{O}(p^2 \log(N))$. Similarly, when updating the $\mathcal{O}(p^2)$ expansion coefficients $L^{\Delta}_{\ell,\alpha_\ell}$ while accepting a move, the number of cells in the interaction list on each level is constant ($6^d - 3^d = 189$

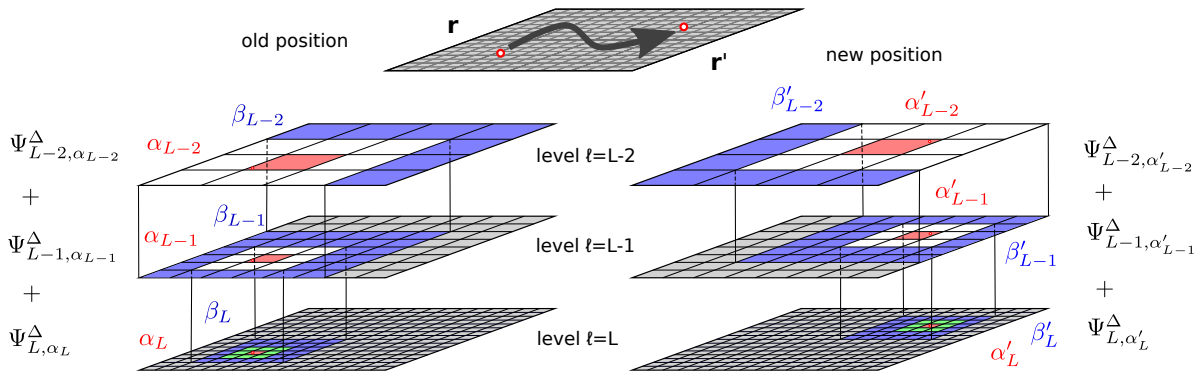

Figure 2: Schematic illustration of the new method for a proposing a MC move $\boldsymbol{r} \to \boldsymbol{r}'$, as described in Section 3.2. On each level ℓ the cell α_ℓ containing $\boldsymbol{r}$ and the corresponding cell α_ℓ containing $\boldsymbol{r}'$ are marked in red. Cells β_ℓ in the interaction list of α_ℓ and cells β'_ℓ in the interaction list of α_ℓ are shown in blue. Interactions with particles in green cells have to be evaluated directly when a move is proposed.

Algorithm L.1 Propose a move $\boldsymbol{r} \to \boldsymbol{r}'$ using local expansions. Input: initial position $\boldsymbol{r}$ of particle with charge q; target position $\boldsymbol{r}'$. Output: change in total electrostatic energy ΔU

Find fine-level cells α_L, α'_L such that $\boldsymbol{r} \in \alpha_L$ and $\boldsymbol{r}' \in \alpha'_L$
$\Delta U \leftarrow 0$
for $\ell = L, L-1, \ldots, 1$ **do**
  Update $\Delta U \leftarrow \Delta U + q\left(\Psi^{\Delta}_{\ell,\alpha'_\ell}(\boldsymbol{r}' - \boldsymbol{R}_{\alpha'_\ell}) - \Psi^{\Delta}_{\ell,\alpha_\ell}(\boldsymbol{r} - \boldsymbol{R}_{\alpha_\ell})\right)$ using *local* expansion in Equation (2)
  if $\ell > 1$ **then**
    Set $\alpha_{\ell-1} \leftarrow \mathsf{parent}(\alpha_\ell)$; $\alpha'_{\ell-1} \leftarrow \mathsf{parent}(\alpha'_\ell)$
  end if
end for
Calculate change ΔU_{direct} in electrostatic energy from direct interactions with Algorithm 4
$\Delta U \leftarrow \Delta U + \Delta U_{\text{direct}}$

Algorithm L.2 Accept a move $\boldsymbol{r} \to \boldsymbol{r}'$ using local expansions. Input: old position $\boldsymbol{r}$, new position $\boldsymbol{r}'$

Find fine-level cells α_L, α'_L such that $\boldsymbol{r} \in \alpha_L$ and $\boldsymbol{r}' \in \alpha'_L$
for $\ell = L, L-1, \ldots, 1$ **do**
  for $\beta_\ell \in \mathrm{IL}(\alpha_\ell)$ and $\beta'_\ell \in \mathrm{IL}(\alpha'_\ell)$ **do**
    Set $(\delta r, \delta\theta, \delta\phi) \leftarrow \mathsf{spherical}(\boldsymbol{r} - \boldsymbol{R}_{\beta_\ell})$ and $(\delta r', \delta\theta', \delta\phi') \leftarrow \mathsf{spherical}(\boldsymbol{r}' - \boldsymbol{R}_{\beta'_\ell})$
    for $n = 0, \ldots, p$ **do**
      for $m = -n, \ldots, +n$ **do**
        Update $(L^{\Delta}_{\ell,\beta_\ell})^m_n \leftarrow (L^{\Delta}_{\ell,\beta_\ell})^m_n - q(\delta r)^{-(n+1)} Y_n^{-m}(\delta\theta, \delta\phi)$
        Update $(L^{\Delta}_{\ell,\beta'_\ell})^m_n \leftarrow (L^{\Delta}_{\ell,\beta'_\ell})^m_n + q(\delta r')^{-(n+1)} Y_n^{-m}(\delta\theta', \delta\phi')$
      end for
    end for
  end for
  if $\ell > 1$ **then**
    Set $\alpha_{\ell-1} \leftarrow \mathsf{parent}(\alpha_\ell)$; $\alpha'_{\ell-1} \leftarrow \mathsf{parent}(\alpha'_\ell)$
  end if
end for

Algorithm L.3 Initialise local expansion coefficients $(L^{\Delta}_{\ell,\alpha})^m_n$ for electrostatic calculation with Algorithms L.1 and L.2

for levels $\ell = 1, \ldots, L$ **do**
  for all cells α_ℓ on level ℓ **do**
    for $n = 0, \ldots, p$ and $m = -n \cdots + n$ **do**
      Set $(L^{\Delta}_{\ell,\alpha_\ell})^m_n = 0$
    end for
  end for
  for all cells α_ℓ on level ℓ **do**
    for all particles with charge q_i and position $\boldsymbol{r}_i \in \alpha_\ell$ **do**
      for all cells $\beta_\ell \in \mathrm{IL}(\alpha_\ell)$ **do**
        Set $(\delta r_i, \delta\theta_i, \delta\phi_i) \mapsto \mathsf{spherical}(\boldsymbol{r}_i - \boldsymbol{R}_{\beta_\ell})$
        for $n = 0, \ldots, p$ **do**
          for $m = -n, \ldots, +n$ **do**
            Update $(L^{\Delta}_{\ell,\beta_\ell})^m_n \mapsto (L^{\Delta}_{\ell,\beta_\ell})^m_n + q_i(\delta r_i)^{-(n+1)} Y^{-m}_n(\delta\theta_i, \delta\phi_i)$
          end for
        end for
      end for
    end for
  end for
end for

Algorithm 4 Calculate change in electrostatic energy from direct interactions for a proposed move $\boldsymbol{r} \to \boldsymbol{r}'$. Input: initial position $\boldsymbol{r} \in \alpha_L$ of particle with charge q; target position $\boldsymbol{r}' \in \alpha'_L$. Output: change in direct electrostatic interaction energy $\Delta U_{\mathrm{direct}}$

$\Delta U_{\mathrm{direct}} \mapsto 0$
for all particles with charge q_i and position $\boldsymbol{r}_i \in \alpha_L \cup \mathcal{N}_b(\alpha_L)$, $\boldsymbol{r}_i \neq \boldsymbol{r}$ **do**
  Update $\Delta U_{\mathrm{direct}} \mapsto \Delta U_{\mathrm{direct}} - \frac{qq_i}{|\boldsymbol{r} - \boldsymbol{r}_i|}$
end for
for all particles with charge q'_i and position $\boldsymbol{r}'_i \in \alpha'_L \cup \mathcal{N}_b(\alpha'_L)$, $\boldsymbol{r}'_i \neq \boldsymbol{r}'$ **do**
  Update $\Delta U_{\mathrm{direct}} \mapsto \Delta U_{\mathrm{direct}} + \frac{qq'_i}{|\boldsymbol{r}' - \boldsymbol{r}'_i|}$
end for
Remove self-interaction $\Delta U_{\mathrm{direct}} \mapsto \Delta U_{\mathrm{direct}} - \frac{q^2}{|\boldsymbol{r}' - \boldsymbol{r}|}$

in $d = 3$ dimensions, to be specific). Therefore the computational complexity of the accept stage is also $\mathcal{O}(p^2 L) = \mathcal{O}(p^2 \log(N))$. The larger constant (compared to the one in the propose stage) arises due to the fact that $2 \times 189 = 378$ instead of 2 cells have to be considered on each level in this stage and is partially compensated by two effects:

1. Typically only a fraction of all proposed moves are accepted.
2. Each MC proposal also requires the calculation of the short-range electrostatic interactions, which is not necessary in the accept stage.

The short-range contribution of the electrostatic potential is evaluated by calculating the contribution of all charges in $\mathcal{N}_b(\alpha)$ and $\mathcal{N}_b(\alpha')$ directly in Algorithm 4. As in the standard FMM algorithm the number of charges per fine level cell is constant and independent of the number of levels; each cell typically contains at the order of 1-10 charges. This guarantees that the total cost of the electrostatic calculation in the proposal step is still $\mathcal{O}(\log(N))$ after the direct, short-range interactions are included using Algorithm 4.

We conclude that the computational complexity of the electrostatics in one MH step, which consists of a proposed move, potentially followed by one accepted transition, is $\mathcal{O}(p^2 \log(N))$.

The initialisation of the local expansions $\Psi^{\Delta}_{\ell,\alpha}$ at the beginning of the simulation could be carried out in $\mathcal{O}(p^4 N)$ time with a minimally modified variant of the standard FMM algorithm, which is written down explicitly as Algorithm 2 in [20]. Apart from renaming $L_{\ell,\alpha} \mapsto L^{\Delta}_{\ell,\alpha}$ in the local expansions, the only difference is that line 17 of this algorithm has to be replaced by $\Psi_{\ell,\alpha} \leftarrow\!\shortmid 0$ and the loop over cells to construct $\overline{\Psi}_{\ell,\alpha}$ in lines 13-15 is no longer necessary. However, since the setup cost is amortised anyway by the large number of MH steps, we chose a slightly more expensive but simpler approach, which is written down in Algorithm L.3 and avoids the calculation of multipole expansions in the upwards pass of the FMM algorithm. For this, the coefficients $L^{\Delta}_{\ell,\alpha}$ are initialised to zero for all cells α and levels ℓ. Next, on each level we loop over all cells α and increment $\Psi^{\Delta}_{\ell,\beta}$ for all $\beta \in \mathrm{IL}(\alpha)$ by adding the contribution of all monopoles in α to the local expansion in β. Since N monopoles have to be considered on each level, the computational complexity of the setup phase is $\mathcal{O}(p^2 L N) = \mathcal{O}(p^2 N \log(N))$.

3.3. Alternative algorithm based on multipole expansions

For reference, we now describe the alternative algorithm introduced in [12], which is based entirely on multipole expansions and which we also implemented for reference. In analogy to Equation (2), we define the multipole expansion of all particles contained in box α on level ℓ around the centre of the box

$$\Phi_{\ell,\alpha}(\boldsymbol{\delta r}) = \sum_{n=0}^{p} \sum_{m=-n}^{+n} (M_{\ell,\alpha})_n^m \, (\delta r)^{-(n+1)} Y_n^m(\delta\theta, \delta\phi) \qquad \text{with } (\delta r, \delta\theta, \delta\phi) = \mathsf{spherical}(\boldsymbol{\delta r}). \tag{3}$$

Assuming that the particles in the box α have coordinates $\boldsymbol{\delta r}_i$ and charges q_i with $i \in I_\alpha \subset \{1, \ldots, N\}$, the explicit expression for the multipole expansion coefficients in Equation (3) is

$$(M_{\ell,\alpha})_n^m = \sum_{i \in I_\alpha} q_i (\delta r_i)^n Y_n^{-m}(\delta\theta_i, \delta\phi_i) \qquad \text{with } (\delta r_i, \delta\theta_i, \delta\phi_i) = \mathsf{spherical}(\boldsymbol{\delta r}_i). \tag{4}$$

Algorithms M.1 and M.2 describe how a potential move is proposed and accepted, using only multipole expansions; the two algorithms should be compared to Algorithms L.1 and L.2 above. Both methods have a computational complexity of $\mathcal{O}(p^2 \log(N))$, but observe that the expensive loops over the interaction list are now carried out in the proposal step in Algorithm M.1. Again, it would be possible to initialise the multipole expansion coefficients $M_{\ell,\alpha}$ at the beginning of the simulation in $\mathcal{O}(p^4 N)$ time with one upward pass of the native FMM method. However, for simplicity we chose to calculate them directly by looping over the cells on all levels and accumulating the multipole coefficients from all particles in a particular cell using Equation (4); this is written down explicitly in Algorithm M.3 which has $\mathcal{O}(p^2 N \log(N))$ complexity. The resulting coefficients $(M_{\ell,\alpha})_n^m$ are not identical to those that would be have been obtained in the upward pass of the FMM algorithm, where they are calculated by recursively combining expansions on subsequent levels. However, the difference between the two ways of computing the multipole coefficients can be bounded as in the standard FMM error analysis.

Algorithm M.1 Propose a move $\boldsymbol{r} \to \boldsymbol{r}'$ using multipole expansions. Input: initial position $\boldsymbol{r}$ of particle with charge q; target position $\boldsymbol{r}'$. Output: change in total electrostatic energy ΔU

Find fine-level cells α_L, α'_L such that $\boldsymbol{r} \in \alpha_L$ and $\boldsymbol{r}' \in \alpha'_L$
$\Delta U \leftarrow\!\shortmid 0$
for $\ell = L, L-1, \ldots, 1$ **do**
  for $\beta_\ell \in \mathrm{IL}(\alpha_\ell)$ and $\beta'_\ell \in \mathrm{IL}(\alpha'_\ell)$ **do**
    Update $\Delta U \leftarrow\!\shortmid \Delta U + q\left(\Phi_{\ell,\beta'_\ell}(\boldsymbol{r}' - \boldsymbol{R}_{\beta'_\ell}) - \Phi_{\ell,\beta_\ell}(\boldsymbol{r} - \boldsymbol{R}_{\beta_\ell})\right)$ using *multipole* expansion in Equation (3)
  end for
  if $\ell > 1$ **then**
    Set $\alpha_{\ell-1} \leftarrow\!\shortmid \mathsf{parent}(\alpha_\ell)$; $\alpha'_{\ell-1} \leftarrow\!\shortmid \mathsf{parent}(\alpha'_\ell)$
  end if
end for
Calculate change $\Delta U_{\mathrm{direct}}$ in electrostatic energy from direct interactions with Algorithm 4
$\Delta U \leftarrow\!\shortmid \Delta U + \Delta U_{\mathrm{direct}}$

Algorithm M.2 Accept a move $\boldsymbol{r} \to \boldsymbol{r}'$ using multipole expansions. Input: old position $\boldsymbol{r}$, new position $\boldsymbol{r}'$

Find fine-level cells α_L, α'_L such that $\boldsymbol{r} \in \alpha_L$ and $\boldsymbol{r}' \in \alpha'_L$
for $\ell = L, L-1, \ldots, 1$ **do**
  Set $(\delta r, \delta\theta, \delta\phi) \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r} - \boldsymbol{R}_{\alpha_\ell})$ and $(\delta r', \delta\theta', \delta\phi') \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r}' - \boldsymbol{R}_{\alpha'_\ell})$
  for $n = 0, \ldots, p$ **do**
    for $m = -n, \ldots, +n$ **do**
      Update $(M_{\ell,\alpha_\ell})_n^m \leftarrow\!\shortmid (M_{\ell,\alpha_\ell})_n^m - q(\delta r)^n Y_n^{-m}(\delta\theta, \delta\phi)$
      Update $(M_{\ell,\alpha'_\ell})_n^m \leftarrow\!\shortmid (M_{\ell,\alpha'_\ell})_n^m + q(\delta r')^n Y_n^{-m}(\delta\theta', \delta\phi')$
    end for
  end for
  if $\ell > 1$ **then**
    Set $\alpha_{\ell-1} \leftarrow\!\shortmid \mathsf{parent}(\alpha_\ell)$; $\alpha'_{\ell-1} \leftarrow\!\shortmid \mathsf{parent}(\alpha'_\ell)$
  end if
end for

Algorithm M.3 Initialise multipole expansion coefficients M_n^m for electrostatic calculation with Algorithms M.1 and M.2

for levels $\ell = 1, \ldots, L$ **do**
  for all cells α_ℓ on level ℓ **do**
    for $n = 0, \ldots, p$ and $m = -n \cdots + n$ **do**
      Set $(M_{\ell,\alpha_\ell})_n^m = 0$
    end for
  end for
  for all cells α_ℓ **do**
    for all particles with charge q_i and position $\boldsymbol{r}_i \in \alpha_\ell$ **do**
      Set $(\delta r_i, \delta\theta_i, \delta\phi_i) \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r}_i - \boldsymbol{R}_{\alpha_\ell})$
      for $n = 0, \ldots, p$ **do**
        for $m = -n, \ldots, +n$ **do**
          Update $(M_{\ell,\alpha_\ell})_n^m \leftarrow\!\shortmid (M_{\ell,\alpha_\ell})_n^m + q_i(\delta r_i)^n Y_n^{-m}(\delta\theta_i, \delta\phi_i)$
        end for
      end for
    end for
  end for
end for

3.4. Boundary conditions

So far we have implicitly assumed that free-space boundary conditions are used for the calculation of the electrostatic energy. In this case the interaction lists on the coarsest two levels of Algorithms L.2, L.3, M.1 and L.3 are empty. This implies that $\Psi^{\Delta}_{1,1} = \Psi^{\Delta}_{2,\alpha} = \Phi_{1,1} = \Phi_{2,\alpha} = 0$ and levels $\ell = 1, 2$ can be skipped when looping over the hierarchical tree. It is possible to adapt all algorithms in this section for simulations with periodic boundary conditions by making the following modifications:

- In Algorithms L.2 and L.3, extend the domain Ω by $3^3 - 1 = 26$ identical copies of the simulation cell to obtain an extended computational domain $\overline{\Omega}$. In the loop over α_ℓ and α'_ℓ, include the copies of those cells in the extended domain $\overline{\Omega}$.

- By following the approach described in detail in [20, Section 3.1], extend Algorithm L.3 to initialise the data structures K and E required to compute the electrostatic contribution of all charges outside $\overline{\Omega}$.

  Set $K_n^m \leftarrow\!\shortmid 0,\ E_n^m \leftarrow\!\shortmid 0\ \forall\ m, n$
  for all particles with charge q_i and position $\boldsymbol{r}_i \in \alpha_\ell$ **do**
    Set $(\delta r_i, \delta\theta_i, \delta\phi_i) \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r}_i - \boldsymbol{R}_1)$
    for $n = 0, \ldots, p$ **do**
      for $m = -n, \ldots, +n$ **do**
        Set $K_n^m \leftarrow\!\shortmid K_n^m + q_i(\delta r_i)^n Y_n^{-m}(\delta\theta_i, \delta\phi_i)$
        Set $E_n^m \leftarrow\!\shortmid E_n^m + q_i(\delta r_i)^n Y_n^{m}(\delta\theta_i, \delta\phi_i)$
      end for
    end for
  end for
  Store K and E.

- Extend Algorithm L.2 to update K and E when a move is accepted.

  Set $(\delta r, \delta\theta, \delta\phi) \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r} - \boldsymbol{R}_1)$ and $(\delta r', \delta\theta', \delta\phi') \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r}' - \boldsymbol{R}_1)$
  for $n = 0, \ldots, p$ **do**
    for $m = -n, \ldots, +n$ **do**
      Set $K_n^m \leftarrow\!\shortmid K_n^m + q\left((\delta r')^n Y_n^{-m}(\delta\theta', \delta\phi') - (\delta r)^n Y_n^{-m}(\delta\theta, \delta\phi)\right)$
      Set $E_n^m \leftarrow\!\shortmid E_n^m + q\left((\delta r')^n Y_n^{m}(\delta\theta', \delta\phi') - (\delta r)^n Y_n^{m}(\delta\theta, \delta\phi)\right)$
    end for
  end for

- Extend Algorithm L.1 to include the contributions to the energy differences from the proposed move in periodic images outside $\overline{\Omega}$ by computing the proposed differences to K and E to determine the change in energy (the linear operator $\mathcal{R}$ is introduced in [20, Section 2.2.1]).

  Set $H \leftarrow\!\shortmid \mathcal{R}(K)$
  Set $U^\infty \leftarrow\!\shortmid \sum_{n=0}^{p} \sum_{m=-n}^{n} E_n^m H_n^m$
  Set $(\delta r, \delta\theta, \delta\phi) \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r} - \boldsymbol{R}_1)$ and $(\delta r', \delta\theta', \delta\phi') \leftarrow\!\shortmid \mathsf{spherical}(\boldsymbol{r}' - \boldsymbol{R}_1)$
  for $n = 0, \ldots, p$ **do**
    for $m = -n, \ldots, +n$ **do**
      Set $\delta K_n^m \leftarrow\!\shortmid K_n^m + q\left((\delta r')^n Y_n^{-m}(\delta\theta', \delta\phi') - (\delta r)^n Y_n^{-m}(\delta\theta, \delta\phi)\right)$
      Set $\delta E_n^m \leftarrow\!\shortmid E_n^m + q\left((\delta r')^n Y_n^{m}(\delta\theta', \delta\phi') - (\delta r)^n Y_n^{m}(\delta\theta, \delta\phi)\right)$
    end for
  end for
  Set $\delta H \leftarrow\!\shortmid \mathcal{R}(\delta K)$
  Set $\Delta U^\infty \leftarrow\!\shortmid \sum_{n=0}^{p} \sum_{m=-n}^{n} \delta E_n^m \delta H_n^m - U^\infty$
  Set $\Delta U \leftarrow\!\shortmid \Delta U + \Delta U^\infty$

- In Algorithm 4, extend $\mathcal{N}_b(\alpha_L)$ and $\mathcal{N}_b(\alpha'_L)$ to include all cells in the extended domain $\overline{\Omega}$. Further, the self-interaction term has to be modified to take into account spurious interactions with the additional

copies of the original particle. As discussed in detail in [20, Section 3.1] this can done by replacing the update in line 8 of Algorithm 4 by

$$\Delta U_{\text{direct}} \mapsto \Delta U_{\text{direct}} - \sum_{\boldsymbol{\nu}\in[-1,0,+1]^3} \frac{q^2}{|\boldsymbol{r}' - (\boldsymbol{r} + a\boldsymbol{\nu})|} + \frac{q^2}{a}\left(6 + \frac{8}{\sqrt{3}} + \frac{12}{\sqrt{2}}\right).$$

Similar modifications have to be made to Algorithms M.1 and M.3. In practice, iteration over the 26 additional copies of the simulation cells can be implemented by modifying data structures such as neighbour lists, see [20] for a more detailed discussion.

4. Implementation

The algorithms described in this paper have been implemented as an extension to the performance portable framework for molecular dynamics (PPMD) described in [13], which is freely available at

https://github.com/ppmd/ppmd

PPMD provides a high-level Python interface for particle-based simulations which require the efficient execution of user-defined operations over all particles or pairs of particles in a system. An obvious example of the latter is the calculation of inter-particle forces in molecular dynamics simulations, but the interface is sufficiently abstract to support more general operations such as the structure analysis algorithms discussed in Section 4 of [13]. PPMD automatically generates efficient code for executing short user-defined C-kernels for all particles or particle-pairs on different parallel architectures; both distributed- and shared- memory parallelism are supported and the code can run on non-standard architectures such as GPUs. Particle-specific data (such as e.g. charge, mass and velocity) is stored as instances of the Python `ParticleDat` class. Particle positions, which contain information that is relevant for parallel domain decompositions, are stored as instances of the specialised `PositionDat` class. In addition to electrostatic potential- and force-calculation with classical Ewald summation [4, 21], PPMD also contains an implementation of the standard FMM algorithm in three dimensions given in [7]. The PPMD framework therefore provides all necessary data structures for storing information (such as local- and multipole- expansion coefficients) on a nested hierarchy of grids which is required to implement the algorithms discussed in this paper. Algorithms L.1 - L.3, M.1 - M.3 and 4 are implemented in the separate `coulomb_mc` Python package which is based on PPMD and can be downloaded from

https://github.com/ppmd/coulomb_mc

Algorithms L.1, L.2, M.1 and M.2 have been implemented as auto-generated C-code. This allows the pre-computation of constant expressions such as combinatorial factors that arise in the evaluation of the spherical harmonics and unrolling of nested loops such as the ones in lines 5-10 of Algorithm L.2 and lines 4-9 in Algorithm M.2. Finally, the generated code is compiled for a specific chip architecture at runtime to ensure optimal performance. Currently our implementation supports cuboid geometries with free space- and periodic boundary conditions.

4.1. FMM-MC user interface

Recall that the local expansion coefficients $L^{\Delta}_{\ell,\alpha}$ which are required to compute (changes of) the electrostatic energy of the system of N particles with charges $\{q_1, q_2, \ldots, q_N\}$ and positions $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N\}$ are initialised with Algorithm L.3. In the `coulomb_mc` package the charges of all particles are represented as a `PositionDat` instance `q`, whereas the positions are stored as a `ParticleDat` object `r`. At the beginning of the simulation the user populates `q` and `r` and uses those to create a `MCFMM_LM` object, which is passed additional information on the domain (such as boundary conditions), the number of levels L in the grid hierarchy and the number of expansion terms. The constructor of the `MCFMM_LM` class then executes Algorithm L.3 to initialise the values of the coefficients $L^{\Delta}_{\ell,\alpha}$. Following this, Algorithm L.1 can be used to compute the change in electrostatic energy which occurs if particle j transitions from its original position $\boldsymbol{r} = \boldsymbol{r}_j$ to a new position $\boldsymbol{r}' = \boldsymbol{r}'_j$. In the code this is realised by calling the `propose()` method of the `MCFMM_LM`

Listing 1: Illustration of how a FMM_MC instance is created and then subsequently used to propose and accept moves.

```
# Create MCFMM_LM instance with positions in PositionDat r and charges in ParticleDat q.
# Here we use 3 tree levels and 12 expansion terms
MC = MCFMM_LM(r, q, domain, 'pbc', r=3, l=12)
# Perform the initial electrostatic solve
MC.initialise()

# Consider move of particle j=7 to new position rj_new = (0.1, 1.0, 5.2)
j = 7
rj_new = np.array((0.1, 1.0, 5.2))

# Propose a move of charge j to position rj_new
dU = MC.propose((j,rj_new))

# Accept a move of charge j to position rj_new
MC.accept((j, rj_new),dU)
```

object and passing it the particle index j and the new position $\boldsymbol{r}' = \boldsymbol{r}'_j$. Finally, Algorithm L.2 updates the local expansions $L^{\Delta}_{\ell,\alpha}$ if the proposed move $\boldsymbol{r} \mapsto \boldsymbol{r}'$ is accepted. Calling the class method `accept()`, which is passed the particle index j and the new position $\boldsymbol{r}'_j$ of particle also executes Algorithm L.2 for the transition $\boldsymbol{r}_j \mapsto \boldsymbol{r}'_j$. It replaces the position of particle j in the `PositionDat r` by $\boldsymbol{r}'_j$ and updates the expansion coefficients $L^{\Delta}_{\ell,\alpha}$

Listing 1 illustrates the creation of an `MCFMM_LM` object, followed by the computation of the change in energy $\texttt{dU} = \Delta U$ which would result from moving particle $j = 7$ to the new position $\texttt{rj\_new} = (0.1, 1.0, 5.2)$ by calling the `propose()` method. In the last line the move to the new position is accepted by calling the `accept()` method. Note that although the example in Listing 1 assumes that the proposed position is identical to the accepted position, this is not the case in general. Because of this and since the code keeps track of the *total* energy of the system at each step, by default the `accept()` method executes Algorithm L.1 to compute the change in system energy ΔU. This can be avoided by passing this change ΔU (which - as shown in Listing 1 - might have been computed in a previous call to `propose()` with the new position that is to be accepted) as an additional parameter to the `accept()` method. The corresponding multipole-based Algorithms M.1 - M.3 can be used by creating an `MCFMM_MM` object which keeps track of the multipole expansion coefficients $M^{\Delta}_{\ell,\alpha}$. The constructor of this class implements Algorithm M.3. The class methods `propose()` and `accept()` implement Algorithms M.1 and M.2 and can be used in exactly the same way as the corresponding methods of the `MCFMM_MM` class described above.

Note that the aim of `coulomb_mc` is to provide functionality for the calculation of (changes in) the electrostatic energy through the high-level `MCFMM_LM` and `MCFMM_MM` classes, which typically dominates the runtime. It is up to the user to implement the overarching Monte Carlo algorithm which generates proposed new positions $\boldsymbol{r}'$ and uses the calculated energy differences to accept or reject particular moves, e.g. in a Metropolis Hastings step.

5. Results

In the following we quantify the performance of the algorithms introduced in Sections 3.2 and 3.3 and implemented as described in Section 4. We demonstrate numerically that, as expected, the time spent in each Monte Carlo step increases logarithmically with the number of particles in the system. To assess its overall performance, we also compare the runtime of our code to version 2.06 of the DL_MONTE package [14, 15] which uses the classical Ewald method to compute electrostatic interactions.

All numerical experiments were carried out on the University of Bath *"Balena"* HPC cluster. Compute nodes of this machine consist of two Intel E5-2650v2 CPUs, and all timing results are reported for sequential

runs on a single core. A snapshot of the source code which can be used to reproduce the results, along with all plotting scripts and raw data is provided at [22]. The code was compiled using version 19.5.281 of the Intel compiler; DL_MONTE was compiled with version 17.1.132 of the same compiler.

5.1. Configuration and Parameter Selection

The accuracy of the algorithms described in Sections 3.2 and 3.3 crucially depends on the number of local/multipole expansion terms, which can be quantified by p, the upper limit in the outer sum in Equations (2) and (3). To provide a fair comparison between the methods introduced in this paper and the Ewald implementation in DL_MONTE, p is adjusted such that acceptance probabilities have errors which are comparable to those in DL_MONTE. For a proposed move $\boldsymbol{r} \rightarrow \boldsymbol{r'}$ which results in a change of energy of ΔU, the relative error in the acceptance probability δP is defined as

$$\delta P = \frac{|P - P^*|}{|P^*|}. \tag{5}$$

Here $P = \exp\left(-\Delta U/(k_B T)\right)$ is the acceptance probability (computed by DL_MONTE or an expansion based method) for a given choice of parameters. Assuming that the exact change in energy is ΔU^*, the exact acceptance probability is denoted as $P^* = \exp\left(-\Delta U^*/(k_B T)\right)$. For a particular move P^* is approximated to high accuracy by computing ΔU^* with the local expansion based method and 26 expansion terms ($p = 25$).

The configuration for our numerical experiments is based on TEST01 [23] in the DL_POLY suite. This setup simulates a simple cubic NaCl crystal of alternating Sodium (Na) and Chloride (Cl) ions with a lattice constant of a =3.3Å. Fully periodic boundary conditions are used for all numerical experiments, which are performed at a temperature of $T = 273$K.

To estimate the relative errors δP in the acceptance probability, we start with an initial configuration of charges which is constructed by creating a cubic lattice of $22 \times 22 \times 22 = 10648$ ions as described in TEST01 and perturbing the initial position of each ion by adding a uniform random shift with a maximum size of $0.01a$ in each spatial direction. Based on this, 1000 moves are proposed (note that no moves are accepted) and for each move the acceptance probabilities P for DL_MONTE and the expansion based approaches are computed along with the "exact" acceptance probability P^*, which is estimated as described above. This process is repeated for 16 different initial configurations to generate a total of 16000 samples for the quantity δP defined in Equation (5).

Figure 3 shows the mean relative error δP (averaged over all 16000 samples) as a function of the number of expansion terms, which varies between 4 and 14 for the expansion based methods. This should be compared to the same quantity computed with DL_MONTE at a fixed solver tolerance of 10^{-6}, indicated by the horizontal dashed line. As those results show, choosing 12 expansion terms ($p = 11$) results in a comparable mean relative error $\overline{\delta P}$ which is smaller than 10^{-3}. Note that for a fixed value of p the local expansion based method (Algorithm L.1) has a slightly higher accuracy than the method which only uses multipole expansions (Algorithm M.1).

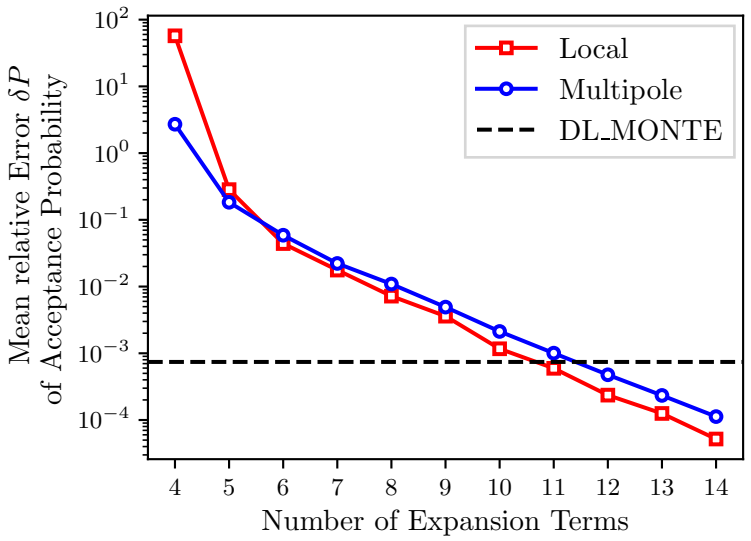

Figure 3: Relative error δP in acceptance probability for DL_MONTE and the expansion based methods for a varying number of expansion terms.

To compare the errors of all used methods in more detail, we also inspect the distribution of δP over all 16000 samples. Figure 4 shows a histogram of the relative error in the acceptance probability, i.e. the number of samples which have a δP that falls into a certain interval $[\delta P_1, \delta P_2]$. Results are shown both for DL_MONTE (again using a solver tolerance of 10^{-6}) and our expansion based methods with 12 expansion terms. The cumulative density of the probability distribution in Figure 4 is plotted in Figure 5. In other words, for a given tolerance ϵ on δP, Figure 5 shows the percentage of samples that have a relative error which does not exceed ϵ. As both figures demonstrate, the spread in errors in slightly larger for the expansion based methods: although for those methods a larger fraction of samples have errors well below the tolerance of 10^{-3}, there is a small number of outliers. This, however, is consistent between the two expansion based methods.

Finally, observe that a large relative error δP in the acceptance probability P will only translate into a large absolute error on P if ΔU^* is also large. It is therefore instructive to also produce a scatter plot of ΔU^* against δP for all samples and this is shown in Figure 6.

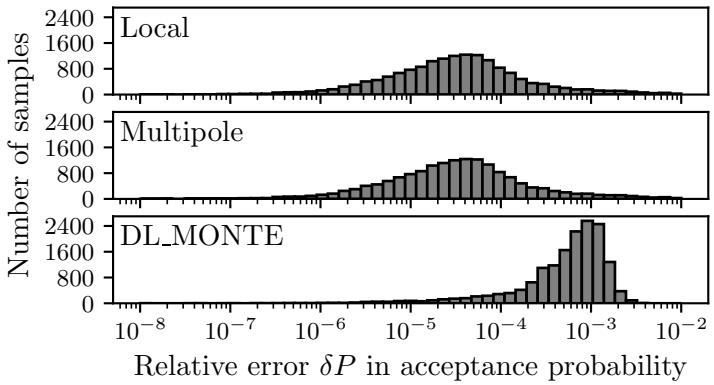

Figure 4: Histograms of relative error δP in acceptance probability. Results are shown for the local expansion based algorithm (top), the multipole expansion based algorithm (middle) and DL_MONTE (bottom); 12 expansion terms ($p = 11$) are used for first two methods.

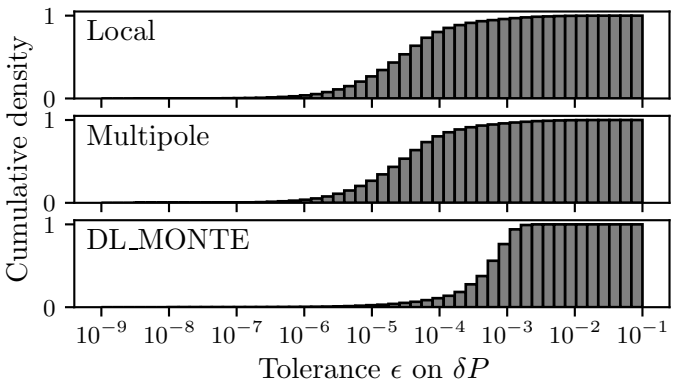

Figure 5: Cumulative density of the relative error δP in computed probabilities. Results are shown for the local expansion based algorithm (top), the multipole expansion based algorithm (middle) and DL_MONTE (bottom); 12 expansion terms ($p = 11$) are used for first two methods.

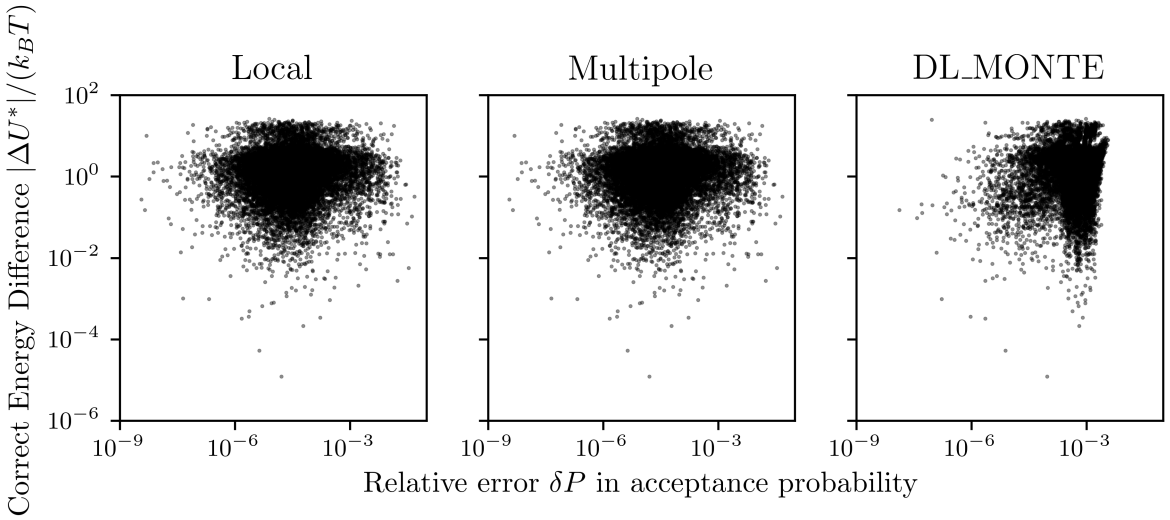

Figure 6: Scatter plot of $\Delta U^*/(k_B T)$ against the relative error δP in acceptance probability. Results are shown for the local expansion based algorithm (left), the multipole expansion based algorithm (middle) and DL_MONTE (right); 12 expansion terms ($p = 11$) are used for first two methods.

5.2. Computational complexity

Next, we investigate the growth in computational cost as a function of the number of charges N. Formally the number of levels L of the hierarchical tree is $\mathcal{O}(\log(N))$. The relative proportion of time spent evaluating the local and multipole expansions in the propose stage (Algorithms L.1 and M.1), update of expansion coefficients (Algorithms L.2 and M.2) and direct, nearest neighbour calculations (Algorithm 4) can be controlled by setting $L = \lfloor \log_8(\alpha N) \rfloor$ and varying the constant α (here $\lfloor \cdot \rfloor$ denotes the floor function defined by $\lfloor x \rfloor = \max\{n \in \mathbb{N} : n \leq x\}$). The optimal value of the parameter α depends on the computer hardware, the average acceptance rate ν and the number of expansion terms. We define the acceptance rate as

$$\nu = \frac{\text{Number of accepted moves}}{\text{Number of proposed moves}}. \tag{6}$$

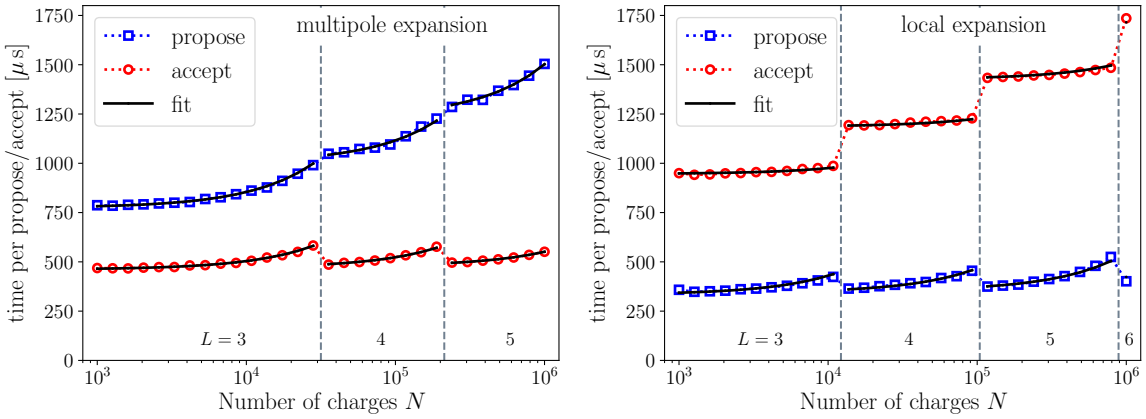

Figure 7: Time spent in the propose and accept operations as a function of the number of charges N in the system. Results in the left figure were obtained with the multipole expansion based method in [12] and show the time spent in Algorithm M.1 ($T^{(\mathrm{mp})}_{\mathrm{prop}}$, blue squares) and Algorithm M.2 ($T^{(\mathrm{mp})}_{\mathrm{acc}}$, red circles). The right figure shows the corresponding numbers for our new method based on local expansion; namely the time spent in Algorithm L.1 ($T^{(\mathrm{loc})}_{\mathrm{prop}}$, blue squares) and Algorithm L.2 ($T^{(\mathrm{loc})}_{\mathrm{acc}}$, red circles). The step-changes in measured times (marked by dashed vertical lines) correspond to increases in the number of levels L, which are shown at the bottom of each figure. The fit to the data (solid black lines) is discussed in Section 5.4.

Assuming that on average $\nu^{-1} = e \approx 2.718$ proposals have to be generated for each accepted move, we find that for our setup and with 12 expansion terms ($p = 11$), the best results are obtained for $\alpha = 0.327$ when using the local expansion based method (Algorithms L.1, L.2) and $\alpha = 0.138$ if the multipole method (Algorithms M.1, M.2) is used. This also implies that for a given value of N, the optimal number of levels for both methods differs by less than 0.5.

To quantify the computational complexity of the propose stage (Algorithms L.1 and M.1) and the accept stage (Algorithms L.2 and M.2) separately, random proposals and accepted moves are generated for problems of increasing size, drawing particle positions and charges from a uniform random distribution with a maximal absolute displacement of 0.25Å in each spatial direction. We investigated the computational cost of proposing and accepting moves for systems containing between a thousand ($N = 10^3$) and a million ($N = 10^6$) particles. For each N the initial arrangement of particles is constructed as described in Section 5.1. Figure 7 shows the average time (measured over 1000 samples) per propose or accept operation as a function of the number N of charges in the system; the fitted solid lines are discussed in Section 5.4 below. The measured times increase abruptly as the number of levels L changes as incrementing the number of levels increases the number of expansions that must be evaluated and updated. Although asymptotically we expect all times to grow as $L \propto \log(N)$, there are significant differences in the rate of growth and absolute computational cost for the different implementations. While for the multipole based method from [12] proposing a single move is significantly more expensive than accepting it, the opposite is true for our new method based on local expansions. The main reason for this is that that the expensive loop over cells in the interaction list has to be executed in the propose stage on the of the multipole based method (Algorithm M.1), whereas the interaction list is traversed in the accept stage of our new method (Algorithm L.2). Overall we therefore expect our new method to be more efficient as the acceptance rate ν decreases, and the number of proposals is significantly larger than the number of accepted moves. In Metropolis Hastings simulations this is the case since the acceptance rate is usually significantly less than 1, a typical value is $1/e \approx 0.3679$.

Although one would naively expect the runtime of the multipole-based accept (Algorithm M.2) and the local-based propose (Algorithm L.1) to be roughly identical, the measured cost of the latter is slightly smaller. Similarly, Figure 7 shows that the multipole-based propose (Algorithm M.1) is slightly faster than the local-based accept (Algorithm L.2). This can be explained by details of the implementation. Firstly, the bookkeeping operations for the propose- and accept stages introduce different overheads. For example, when

proposing a move the cells α_ℓ, α'_ℓ have to be computed and when a move is accepted data structures, such as the indirection map that assigns particles to cells, must be updated. Furthermore, in practice reading and writing floating point numbers carries a different cost. For example, in the loop over the interaction list in Algorithm L.2 expansion coefficients are updated, which requires a read and a write operation, however in Algorithm M.1 in the loop over the interaction list expansion coefficients are only read, which avoids any potential write contention. Finally bear in mind that the simulation uses periodic boundary conditions. In our implementation this introduces a small overhead when proposing a move and a slightly larger additional cost when a move is accepted.

To understand the growth of the runtime for increasing problem size N in more detail observe the following: when an additional level is added to the octal tree as N increases, initially there are $N/N_{\text{cell}} = \mathcal{O}(1)$ charges per cell on the finest level. However, as the total number of charges increases while the number of levels L is kept fixed, the average number of charges per cell will grow. Hence the cost of Algorithms L.1 and M.1 grows in proportion to N/N_{cell} for fixed L since both require the computation of direct interactions with Algorithm 4. In addition our implementation of Algorithms L.2 and L.2 contains a small number of bookkeeping operations which are formally $\mathcal{O}(N)$. As discussed in Section 5.4, the absolute contribution of those operations to the total runtime is very small (and will be amoritised by the cost of direct interactions induced by other short-range potentials such a repulsive Lennard-Jones field in real-life simulations). Ignoring this small $\mathcal{O}(N)$ contribution the asymptotic complexity of Algorithms L.1, L.2, M.1 and M.2 is $\mathcal{O}(L) = \mathcal{O}(\log(N))$.

5.3. Comparison of full Monte Carlo simulation with DL_MONTE

Having quantified the time spent in the propose- and accept-stage of our expansion based algorithm separately, we now discuss the growth of the total runtime of an entire Monte Carlo simulation as a function of the problem size. For this we compare the performance of our expansion-based implementations in PPMD against DL_MONTE. Again we consider an NaCl crystal with the same initial arrangement of charges as described in Section 5.1. To prevent oppositely charged ions from collapsing onto one another over the course of the simulation, a repulsive short-range Lennard-Jones potential with a fixed cutoff of 12Å is added. For the largest problem sizes (with $N \approx 10^5$ in a cubic box with a side length of 153.2Å) this short-range potential adds an additional average cost of 0.18ms per Monte Carlo step for our expansion based methods. This accounts for approximately 14% of the total average cost per step for the local expansion implementation.

For each proposed move we create a random offset vector $\delta\boldsymbol{r} = \boldsymbol{r}' - \boldsymbol{r}$, such that each component δr_j is uniformly distributed in the interval $[-0.25\text{Å}, +0.25\text{Å}]$. This resulted in an average acceptance rate of $\nu \approx 0.438$ for the expansion based methods. Note that the performance of the Ewald implementation in DL_MONTE is not sensitive to the acceptance rate and that in DL_MONTE the additional cost of accepting a proposed move is negligible.

Figure 8 shows the time per MC step for our implementations of the expansion based methods and for DL_MONTE as a function of the number of charges N. The size of the system varies between $N = 10^3$ and $N = 10^5$ charges and the reported times are averaged over 1000 Metropolis-Hastings steps (i.e. 1000 proposed moves). We do not include the setup times, since those account for an insignificant fraction of the runtime for "real" simulations with a large number of Metropolis-Hastings steps.

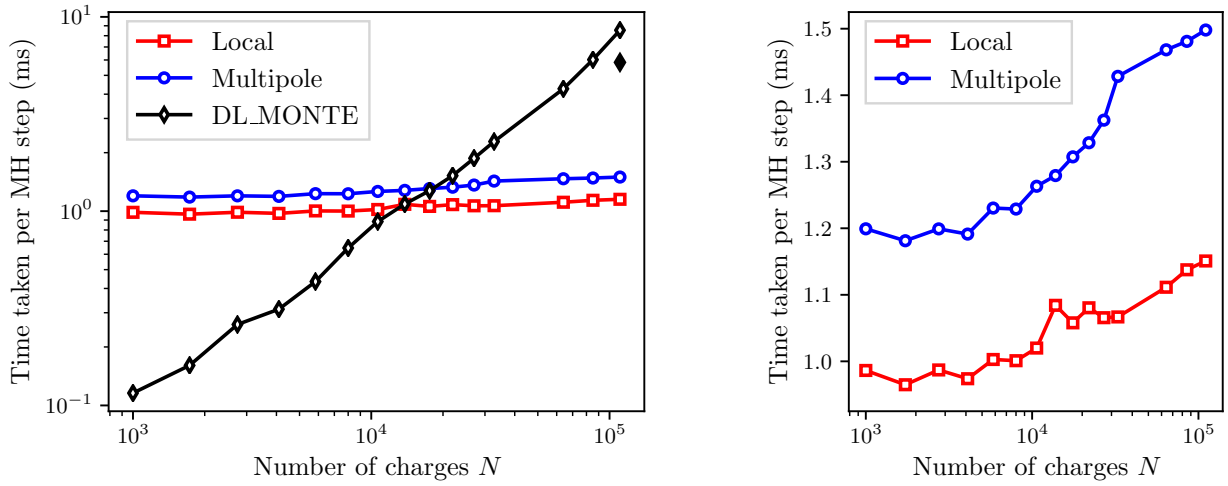

Figure 8: Time per Metropolis Hastings step for our implementations of the expansion based methods and DL_MONTE. In the latter case the cutoff was kept fixed at $r_c = 12$Å for all problem sizes, except for the rightmost data point where results with an optimised cutoff of $r_c = 28$Å are also shown as a solid diamond. The plot on the right shows the same data but with a linear scale on the vertical axis.

For $N < 10^4$ particles DL_MONTE provides more performance than our implementations, but it is almost an order magnitude slower for the largest considered problem size ($N = 10^5$). Empirically the cost of the expansion based methods grows only relatively slowly with the problem size. For DL_MONTE, on the other hand, the runtime is approximately proportional to the number of particles. This $\mathcal{O}(N)$ growth can be explained as follows: for the direct Ewald summation method with a short-range cutoff r_c, short-range interactions with $\mathcal{O}(\rho r_c^3)$ particles have to be computed for each proposed move in a system with density ρ. To balance errors in the short- and long-range computation, the physical cutoff in Fourier space and r_c are inversely related. As a consequence, the number of Fourier modes that have to be considered grows like $\mathcal{O}(V r_c^{-3})$. Hence, for fixed density where $V = \rho N$ the combined cost of the short- and long- range contributions is

$$\mathrm{Cost}_{\mathrm{Ewald}}(r_c, N) = \mathcal{O}(r_c^3 + N r_c^{-3}). \tag{7}$$

This cost can be minimised by varying the cutoff such that $r_c = r_c^{(0)} N^{1/6}$ for some constant $r_c^{(0)}$, as discussed for example in [24] (see also [17, Chapter 12]), resulting in a total cost of $\mathrm{Cost}_{\mathrm{Ewald}}(r_c^{(0)} N^{1/6}, N) = \mathcal{O}(N^{1/2})$ per proposal for the Ewald method. In the current version of DL_MONTE the short-range cutoff for the Ewald summation has to be identical to the cutoff for Lennard-Jones interactions, which we fixed at $r_c = 12$Å to obtain the majority of the results in Figure 8. In addition we also quantify how the results would change if the optimal short range cutoff is chosen. At the moment this requires manual fine-tuning and for practical reasons it was not possible to do this for all problem sizes. Equation (7) shows that for a fixed cutoff of $r_c = 12$Å the cost of the Ewald summation is dominated by the long range contribution and grows in proportion to N. While this is a current limitation of DL_MONTE and not a fundamental issue, it is worth stressing that even if this problem is overcome, Ewald summation has a computational complexity of $\mathcal{O}(N^{1/2})$ compared to the $\mathcal{O}(\log(N))$ complexity of our hierarchical methods. To demonstrate the effect of a more optimal short-range cutoff we varied the cutoff radius r_c between 12Å and 32Å and found that a cutoff of $r_c = 28$Å gives near optimal results. As shown by the rightmost datapoints in Figure 8, where the results obtained with $r_c = 28$Å are indicated by a solid diamond, this reduces the time per MH step from 8.0ms to 5.8ms for $N \approx 10^5$ charges. This value might be reduced further in future releases of DL_MONTE, if the different cutoffs can be varied independently to avoid the evaluation of short range potentials with an unnaturally large cutoff. With an optimally tuned cutoff the Ewald crossover between the Ewald- based method and our hierarchical algorithms would occur for larger problem sizes.

expansion	stage	τ	γ	κ	σ
multipole	propose	89.15	228.40	0.50	$1.49 \cdot 10^{-4}$
	accept	436.71	8.12	0.27	$7.11 \cdot 10^{-6}$
local	propose	304.01	9.92	0.61	$4.12 \cdot 10^{-5}$
	accept	225.80	239.99	0.19	$4.37 \cdot 10^{-5}$

Table 1: Numerical values of fit parameters for empirical performance model in Equation (8).

Finally, note that in its current implementation the setup cost of DL_MONTE grows like $\mathcal{O}(N^2)$ instead of $\mathcal{O}(N^{3/2})$ which could be achieved for an optimal $r_c \propto N^{1/6}$. Although not considered here, this $\mathcal{O}(N^2)$ setup time can become computationally significant for systems containing a large number of charges.

5.4. Empirical results for growth of runtime

As will be shown below, the average speedup of our method relative to the one in [12] is 1.25 over all considered problem sizes. The smallest measured speedup is 1.18, while the maximal speedup is 1.34, with a median value of 1.23. To model the speedup for large N and establish an upper limit on the performance gains as $N \to \infty$ we fit the measured times in Figure 7 to

$$T(N) = \tau + \gamma L + \kappa \frac{N}{N_{\text{cell}}} + \sigma N \tag{8}$$

with a least squares approach. In this expression L is the number of levels of the octree hierarchy for a given value of N and $N_{\text{cell}} = 8^{L-1}$ is the number of cells on the finest level for this octree depth. The numerical values of the four fit parameters τ, γ, κ and σ for the propose- and accept stages of the multipole- and local expansion based methods are given in Table 1. As the solid curves in Figure 7 show, Equation (8) models the data extremely well. As expected the coefficient of the $\mathcal{O}(N)$ term is very small. The empirical model in Eq. (8) allows to predict the expected speedup for a given problem size N and acceptance rate ν as

$$S(N) = \frac{T^{(\text{mp})}_{\text{prop}}(N) + \nu T^{(\text{mp})}_{\text{acc}}(N)}{T^{(\text{loc})}_{\text{prop}}(N) + \nu T^{(\text{loc})}_{\text{acc}}(N)} \tag{9}$$

where superscripts "(loc)" and "(mp)" denote the local and multipole expansion based methods and subscripts "prop" and "acc" label propose and accept operations respectively. Ignoring the $\mathcal{O}(N)$ term in Equation (8) (this can be justified by the small value of σ in Table 1), the asymptotic speedup is

$$S_\infty = \lim_{N\to\infty} S(N) = \frac{\gamma^{(\text{mp})}_{\text{prop}} + \nu\gamma^{(\text{mp})}_{\text{acc}}}{\gamma^{(\text{loc})}_{\text{prop}} + \nu\gamma^{(\text{loc})}_{\text{acc}}} \tag{10}$$

Further observe that asymptotically the relative cost of Algorithms L.1 and M.1 is $\gamma^{(\text{mp})}_{\text{prop}}/\gamma^{(\text{loc})}_{\text{prop}} = 23.03$ while the relative time spent in Algorithms M.2 and L.2 is $\gamma^{(\text{loc})}_{\text{acc}}/\gamma^{(\text{mp})}_{\text{acc}} = 29.56$. These ratios deviate from the theoretically expected value of 189 (which, as the reader will recall, is the number of cells in the interaction list) due to details of the implementation. In particular looping over a larger set of cells in the interaction lists allows the compiler to carry out additional optimisation and vectorise the code.

In Figure 9 we plot the theoretical and measured speedup for an acceptance rate $\nu = 0.438$. Here the theoretical speedups (blue) predicted with Equation (8) are computed from the time per propose and accept operation computed in Section 5.2 and fitted with the model in Equation (8). The measured speedups (red) are obtained from the data visualised in Figure 8. The theoretical upper limit S_∞ for the speedup defined in Equation (10) is also shown; for the fitted values of γ given in Table 1 and with $\nu = 0.438$, we find $S_\infty = 2.02$. As Figure 9 demonstrates, the fit reproduces the measured speedup, which lies in the range $[1.18, 1.34]$, reasonably well. The expected asymptotic speedup for large N is around 2, demonstrating that our method has the potential to approximately halve the runtime compared to the algorithm in [12]. However, this speedup is only reached for values of N which are significantly larger than the ones considered in this work; for $N = 10^6$ we expect a speedup of roughly 1.5×.

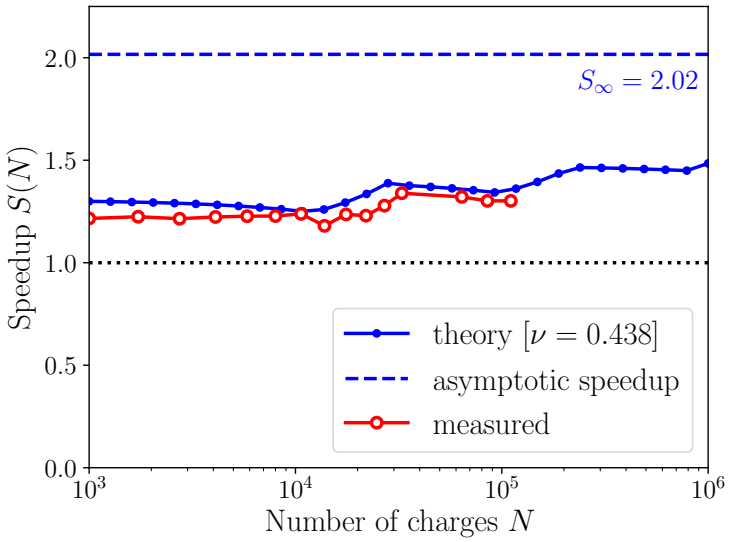

Figure 9: Theoretical speedup $S(N)$ (solid blue line) as defined in Equations (8) and (9) for $\nu = 0.438$; the asymptotic speedup S_∞ (dash blue line), is given in Equation (10). The measured speedup, in solid red, is computed from the local and multipole timings plotted in Figure 8.

6. Conclusion

In this paper we presented a new hierarchical method for accurately including electrostatic interactions in Monte Carlo simulations. Our algorithm has a computational complexity of $\mathcal{O}(\log(N))$ per Metropolis Hastings step. Compared to the related technique in [12], our method reduces the average cost of a MH step as the balance of work between the propose and accept operations is more favourable for typical acceptance rates. Numerically we find that runtimes are reduced by around 30% for systems with $N = 10^5$ charges, with the potential of a speedup of around $2\times$ for larger values of N. We demonstrated numerically that our implementation will effectively scale to systems containing 10^6 charges whilst maintaining the expected computational complexity of $\mathcal{O}(\log(N))$ per MH step. As the direct Ewald summation technique has a higher complexity of at least $\mathcal{O}(\sqrt{N})$, our implementation becomes more efficient for simulations with more than 10^4 particles: for $N = 10^5$ it is about an order of magnitude faster than the current DL_MONTE implementation.

There are several avenues for future work. One obvious shortcoming of the present implementation is the lack of parallelisation. While Monte Carlo simulations are "embarrassingly parallel", and several Markov Chains can be generated in parallel without communications, this increases memory requirements. In our method this issue could be reduced to a certain degree by parallelising over the levels in the grid hierarchy. This is possible since the cost on each level is constant, and computations can be carried out independently, before summing the total energy in the propose stage.

Furthermore, here we have only considered single-particle moves and future work should also investigate other Monte Carlo transitions. For example, in a canonical ensemble the pressure is fixed but the volume of the simulation cell varies. In this case a proposed move could consist of a change of the system volume. In the worst case scenario, the energy change of such a proposed volume move is computed with a full solve of the electrostatic energy of the proposed state. When using an Ewald based approach this system solve incurs an $\mathcal{O}(N^{\frac{3}{2}})$ cost per MH step, however, with multipole- or local expansion based approaches this can potentially be reduced to an $\mathcal{O}(N)$ cost.

Acknowledgements

This research made use of the Balena High Performance Computing (HPC) Service at the University of Bath. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 646176 and No 824158.

[1] G. A. Cisneros, M. Karttunen, P. Ren, C. Sagui, Classical electrostatics for biomolecular simulations, Chemical reviews 114 (1) (2013) 779–814. doi:10.1021/cr300461d.

[2] M. Jorge, N. A. Seaton, Long-range interactions in Monte Carlo simulation of confined water, Molecular Physics 100 (13) (2002) 2017–2023. doi:10.1080/00268970110099585.

[3] C. K. Sandalci, C. Koc, S. M. Goodnick, Three-dimensional Monte Carlo device simulation with parallel multigrid solver, International Journal of High Speed Computing 9 (03) (1997) 223–236. doi:10.1142/S0129053397000143.

[4] P. P. Ewald, Die Berechnung optischer und elektrostatischer Gitterpotentiale, Annalen der Physik 369 (3) (1921) 253–287. doi:10.1002/andp.19213690304.

[5] L. Greengard, V. Rokhlin, A Fast Algorithm for Particle Simulations, Journal of Computational Physics 73 (2) (1987) 325–348. doi:10.1016/0021-9991(87)90140-9.

[6] L. Greengard, The rapid evaluation of potential fields in particle systems, MIT press, 1988.

[7] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta numerica 6 (1997) 229–269. doi:10.1017/S0962492900002725.

[8] J. C. Crabtree, M. Molinari, S. C. Parker, J. A. Purton, Simulation of the Adsorption and Transport of CO2 on Faujasite Surfaces, The Journal of Physical Chemistry C 117 (42) (2013) 21778–21787. doi:10.1021/jp4053727.

[9] T. Pham, B. Space, Insights into the Gas Adsorption Mechanisms in Metal–Organic Frameworks from Classical Molecular Simulations., Top Curr Chem 378 (2020) 14–. doi:10.1007/s41061-019-0276-x.

[10] A. D. Bruce, A. N. Jackson, G. J. Ackland, N. B. Wilding, Lattice-switch Monte Carlo method, Phys. Rev. E 61 (2000) 906–919. doi:10.1103/PhysRevE.61.906.
URL https://link.aps.org/doi/10.1103/PhysRevE.61.906

[11] T. L. Underwood, G. J. Ackland, Lattice-switch Monte Carlo: the fcc—bcc problem, Journal of Physics: Conference Series 640 (2015) 012030. doi:10.1088/1742-6596/640/1/012030.
URL https://doi.org/10.1088%2F1742-6596%2F640%2F1%2F012030

[12] T. A. Höft, B. K. Alpert, Fast updating multipole Coulombic potential calculation, SIAM Journal on Scientific Computing 39 (3) (2017) A1038–A1061. doi:10.1137/16M1096189.

[13] W. R. Saunders, J. Grant, E. H. Müller, A domain specific language for performance portable molecular dynamics algorithms, Computer Physics Communications 224 (2018) 119–135. doi:10.1016/j.cpc.2017.11.006.

[14] J. Purton, J. C. Crabtree, S. Parker, DL_MONTE: a general purpose program for parallel Monte Carlo simulation, Molecular Simulation 39 (14-15) (2013) 1240–1252. doi:10.1080/08927022.2013.839871.

[15] A. V. Brukhno, J. Grant, T. L. Underwood, K. Stratford, S. C. Parker, J. A. Purton, N. B. Wilding, DL_MONTE: a multipurpose code for Monte Carlo simulation, Molecular Simulation (2019) 1–21doi:10.1080/08927022.2019.1569760.

[16] Z. Gan, Z. Xu, Efficient implementation of the Barnes-Hut octree algorithm for Monte Carlo simulations of charged systems, Science China Mathematics 57 (7) (2014) 1331–1340. doi:10.1007/s11425-014-4783-5.

[17] D. Frenkel, B. Smit, Understanding molecular simulation: From algorithms to applications, Vol. 1, Academic press, San Diego/London, 2001.

[18] U. Trottenberg, C. W. Oosterlee, A. Schuller, Multigrid, Academic Press, London, 2001.

[19] J. Rottler, A. C. Maggs, A continuum, $\mathcal{O}(N)$ Monte Carlo algorithm for charged particles, The Journal of chemical physics 120 (7) (2004) 3119–3129. doi:10.1063/1.1642590.

[20] W. R. Saunders, J. Grant, E. H. Müller, I. Thompson, Fast electrostatic solvers for kinetic Monte Carlo simulations, Journal of Computational Physics (2020) 109379doi:10.1016/j.jcp.2020.109379.

[21] W. R. Saunders, J. Grant, E. H. Müller, Long Range Forces in a Performance Portable Molecular Dynamics Framework, in: Parallel Computing is Everywhere, 2018, pp. 37 – 46. doi:10.3233/978-1-61499-843-3-37.

[22] W. R. Saunders, J. Grant, E. H. Mueller, Code and Data Release: Monte Carlo simulations with fast and accurate electrostatics (Jun. 2020). doi:10.5281/zenodo.3873307.
URL https://doi.org/10.5281/zenodo.3873307

[23] CCP5, DL_POLY_4 TEST01, ftp://ftp.dl.ac.uk/ccp5/DL_POLY/DL_POLY_4.0/DATA/, [Online; accessed 01/04/2018] (2018).

[24] J. Kolafa, J. W. Perram, Cutoff errors in the Ewald summation formulae for point charge systems, Molecular Simulation 9 (5) (1992) 351–368. doi:10.1080/08927029208049126.