Fast electrostatic solvers for kinetic Monte Carlo simulations

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

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

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

Abstract

Kinetic Monte Carlo (KMC) is an important computational tool in theoretical physics and chemistry. In contrast to standard Monte Carlo, KMC permits the description of time dependent dynamical processes and is not restricted to systems in equilibrium. Compared to Molecular Dynamics, it allows simulations over significantly longer timescales. Recently KMC has been applied successfully in modelling of novel energy materials such as Lithium-ion batteries and organic/perovskite solar cells. Motivated by this, we consider general solid state systems which contain free, interacting particles which can hop between localised sites in the material. The KMC transition rates for those hops depend on the change in total potential energy of the system. For charged particles this requires the frequent calculation of electrostatic interactions, which is usually the bottleneck of the simulation. To avoid this issue and obtain results in reasonable times, many studies replace the long-range potential by a phenomenological short range approximation. This, however, leads to systematic errors and unphysical results. On the other hand standard electrostatic solvers such as Ewald summation or fast Poisson solvers are highly inefficient in the KMC setup or introduce uncontrollable systematic errors at high resolution.

In this paper we describe a new variant of the Fast Multipole Method by Greengard and Rokhlin which overcomes this issue by dramatically reducing computational costs. We exploit the fact that each update in the transition rate calculation corresponds to a single particle move and changes the configuration only by a small amount. This allows us to construct an algorithm which scales linearly in the number of charges for each KMC step, something which had not been deemed to be possible before.

We demonstrate the performance and parallel scalability of the method by implementing it in a performance portable software library, which was recently developed in our group. We describe the high-level Python interface of the code which makes it easy to adapt to specific use cases.

Keywords: kinetic Monte Carlo, electrostatics, Fast Multipole Method, parallel computing, Domain Specific Language

1. Introduction

The kinetic Monte Carlo (KMC) method [1, 2, 3, 4] was originally developed for the simulation of time dependent statistical processes in chemical reaction dynamics. More recently, the method has been applied in computational physics and chemistry to model processes on grain surfaces [5], in electrolytes [6] and organic devices [7, 8]. In contrast to standard Monte Carlo methods, such as the Metropolis Hastings algorithm [9, 10] for the simulation of Markov processes, KMC is not limited to systems in equilibrium. Instead, it allows the representation of dynamical processes in physical materials, while not being limited by the restrictive timestep constraints in Molecular Dynamics (MD) simulations which arise due to fast, but dynamically irrelevant, oscillations around semi-stable configurations.

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

Kinetic Monte Carlo for energy materials. An important application of the KMC method, which is currently attracting significant interest, is the simulation of transport processes in energy materials. This includes solid state electrolytes such as Lithium-ion batteries [6] and semiconductors in full-device simulations [7, 8]. A particularly promising application are organic- and perovskite- based solar cells, which can achieve remarkable power efficiencies [11, 12]. All those applications require the modelling of dynamic transport processes in a three dimensional volume to predict material properties such charge mobilities and the current-voltage characteristics.

In addition to making direct physical predictions, KMC simulations are also important to adjust parameters in large-scale drift-diffusion models via upscaling. Continuum models of this type are widely used in industry for full-device simulations.

The dynamics of a wide class of systems can be described by hopping processes of particles between sites of a static background matrix. To resolve physically relevant macroscopic features, such as grain boundaries, the simulation domain has to be large and simulated systems have to contain $N = 10^3 - 10^6$ particles and hopping sites. The hopping rates (commonly referred to as "propensities" in the KMC literature), which serve as an input to the KMC algorithm, depend on the total potential energy of the system. This energy is given by the sum of classical interaction potentials for all particle pairs.

Electrostatic interactions. For charged particles the electrostatic contribution to the inter-particle potential is long-range. The Coulomb interactions between all particles need to be computed, resulting naively in an expensive $\mathcal{O}(N^2)$ calculation *per potential hopping event*. This computational complexity can be reduced to $\mathcal{O}(N^{3/2})$ with Ewald summation [13] and $\mathcal{O}(N \log(N))$ with particle mesh methods [14, 15]. However, even if the Fast Multipole Method (FMM) [16, 17, 18] is used (in its standard form), the computational complexity *per hop* is still $\mathcal{O}(N)$. Since $\mathcal{O}(N)$ potential hopping events have to be considered in each KMC step, the total cost per step is at least $\mathcal{O}(N^2)$. Because of this quadratic growth in complexity, the electrostatic calculation is typically the bottleneck of the KMC simulation. As argued in [19], this appears to make large-scale KMC simulations with accurate electrostatics computationally infeasible in principle. To overcome this issue, the Coulomb potential is often replaced by an ad-hoc truncated short range interaction, see e.g. [6]. Since only the interactions with a small number of neighbours need to be calculated in this case, the cost of each potential hopping event is reduced to $\mathcal{O}(1)$, resulting in a total cost of $\mathcal{O}(N)$ per KMC step. However, this truncation introduces uncontrollable systematic errors, which limit the predictive power of the model [20]. For example, the authors of [21] find that neglecting long range interactions when modelling protonic diffusion and conduction in doped perovskites changes the predicted diffusion coefficient by 14% compared to the "correct" results obtained with the very expensive Ewald method. As we will discuss below, other approximation methods such as mapping the charges to a grid and solving the Poisson equation [22], possibly in lower dimensions [23, 24], also introduces uncontrollable systematic errors.

An algorithmically optimal Fast Multipole Method. In this paper we introduce a modification of FMM for KMC. We show that this overcomes the fundamental issues described in [19]. With our modified FMM algorithm the cost per KMC step grows linearly in the number of charges (and not quadratically as claimed in [19]) and accurate electrostatic interactions can be included in large-scale KMC simulations. The key observation is that - since FMM describes the long-range contribution as a continuous field - the change in the electrostatic potential energy can be evaluated at a cost of $\mathcal{O}(1)$ (i.e. independent of the particle number) for each proposed hopping event. As there are $\mathcal{O}(N)$ potential hopping events per KMC step, the total computational complexity of the propensity calculation is $\mathcal{O}(N)$. Updating the FMM field after one hop is accepted carries an additional cost of $\mathcal{O}(N)$, resulting in a total computational complexity per KMC step which scales linearly in the number of charges.

While not the topic of this paper, we remark that it is also possible to improve the computational complexity of *standard* Monte Carlo (MC) by similar methods. As will be argued at the end of this paper, we believe that changes to the electrostatic energy for each *individual attempted MC move* can be calculated at a computational cost $\mathcal{O}(\log(N))$ with a suitably modified version of FMM.

To simulate large physical systems, an efficient, parallel implementation of the algorithm is important to obtain meaningful results in a reasonable time. Easy integration into existing simulation packages and

workflows can be achieved by providing a minimal yet flexible user-interface. With the recent diversification of the hardware landscape, the code should be performance portable and run on different chip architectures, including, for example, traditional CPUs and GPUs. The implementation described in this paper is based on the performance portable framework first introduced in [25]. By providing a Python interface and using code generation techniques, the code is fast, yet allows the user to express their algorithms at a high abstraction level.

For an idealised setup we find that our FMM-KMC algorithm can be used to carry out simulations with exact electrostatics on problems with 10^6 charges in 0.14s per KMC step when running on a parallel computer with 8192 cores. In a physically realistic configuration the hopping processes of 20412 particles in a α-NPD problem doped with F6TCNNQ at a concentration of 2% could be simulated at a rate of 0.35s per KMC step on a single 12-core Skylake CPU.

Structure. This paper is organised as follows: After reviewing the key concepts of KMC and FMM in Section 2 we describe our adaptation of the FMM algorithm for KMC simulations in Section 3 and review related work in Section 4. An efficient implementation of our method based on the performance portable framework in [25] is described in Section 5, where we discuss the user-interface in detail. Numerical results which demonstrate the accuracy, computational efficiency and parallel scalability of the algorithm for idealised model systems and a physically relevant setup are presented in Section 6. Finally, we conclude and discuss possible future directions of our work in Section 7. Some more technical aspects are relegated to the appendices. The standard FMM algorithm is written down in Appendix A and the correction term for charge distributions with a non-vanishing dipole-moment is derived in Appendix B. An improved user interface for proposing moves, which is optimised for efficiency, is described in Appendix C. Previously, we reported on the performance of Ewald-based long range electrostatics in the same code base [26]. To complement this work we discuss the performance and scalability of the standard FMM implementation in Appendix D.

2. Review of Methods

To put our new algorithm into context and establish necessary notation, we first review the KMC method and the standard FMM algorithm.

2.1. Kinetic Monte Carlo

Molecular Dynamics (MD) and Monte Carlo (MC) are the standard computational tools for predicting the properties of physical and chemical systems from first principles (see e.g. [27, 28]). Typically it is assumed that the molecular constituents interact via phenomenological classical potentials; for charged systems this includes long range electrostatic interactions. While MC can be used to study systems in equilibrium by sampling from the steady state distribution, MD allows the simulation of dynamical processes such as time dependent charge propagation in batteries and solar cells. In solid state systems at moderate temperatures and pressures there are often two types of processes which occur at very different time scales: fast oscillations around local minima of the energy landscape, which are separated by large energy barriers, and much slower transitions between those minima. In a system with these properties MD is highly inefficient for extracting quantities such as charge mobilities and voltage characteristics. This is because the MD timestep needs to be small enough to resolve the fast oscillations, yet the trajectories have to be sufficiently long to include dynamically relevant transitions between local minima. In fact, the rapid oscillations do not contain interesting physical information on charge transport processes, and should be integrated out. Kinetic Monte Carlo (KMC) [1, 2, 3, 4] overcomes this problem by treating each of the local minima as an independent configuration or state $S_{\mathfrak{a}}$ (here and in the following indices $\mathfrak{a}, \mathfrak{b}, \ldots$ are used to label states; particles are indexed with Roman letters $i, j, \ldots$ and Greek letters $\alpha, \beta, \ldots$ are used for cells in the computational grid). The dynamics are approximated by probabilistic transitions between those configurations. The generated probabilistic trajectory is equivalent to snapshots of the full MD simulation at discrete times. For example, free charge carriers in a crystal at room temperature are bound to specific local sites, and the states correspond to particular distributions of the particles, such that every site is either empty or occupied. KMC

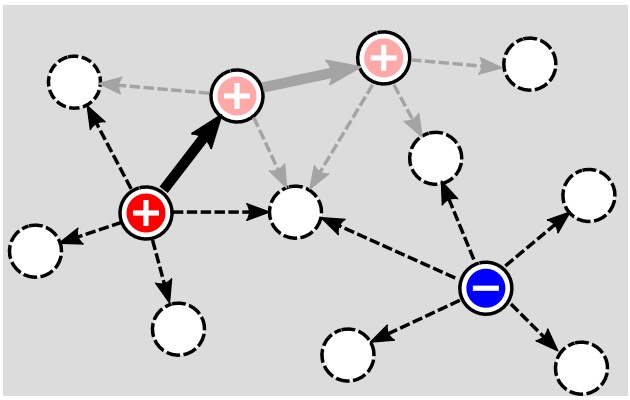

Figure 1: Schematic sketch of one KMC step, which consists of calculation of the propensities for all possible hops (dashed arrows) and moving one particle to a new site after accepting a particular hop (solid arrows). Subsequent steps are shown as gray arrows.

assumes that there is a fixed rate $r_{\mathfrak{a}\mathfrak{b}}$ for a configuration to transition from state $S_{\mathfrak{a}}$ to state $S_{\mathfrak{b}}$ in a given time, i.e. each transition is modelled as a Poisson process. The rates $r_{\mathfrak{a}\mathfrak{b}}$ are known as "propensities" in the KMC literature. This is a valid approximation if we assume that - compared to the state-transition time scale - the fast oscillations around the local equilibria occur so rapidly that the particles "forget" their previous history, and the transition probability is the same at each point in time. The propensities are inputs to the KMC algorithm and it typically assumed that $r_{\mathfrak{a}\mathfrak{b}}$ depends on the energy difference between states $S_{\mathfrak{a}}$ and $S_{\mathfrak{b}}$. Since transitions between states are modelled by a Poisson process, the probability distribution function for the time of the first escape from the state $S_{\mathfrak{a}}$ is

$$\pi_{\mathfrak{a}}(t) = Q_{\mathfrak{a}} e^{-Q_{\mathfrak{a}} t} \qquad \text{where } Q_{\mathfrak{a}} := \sum_{\mathfrak{b}=1}^{\mathfrak{N}} r_{\mathfrak{a}\mathfrak{b}}. \tag{1}$$

Here $\mathfrak{N}$ is the total number of states. This information is used to work out the physical time interval between subsequent snapshots. The following two steps occur at each transition between states:

1. Starting from state $S^{(n)} = S_{\mathfrak{a}}$, pick a new state $S^{(n+1)} = S_{\mathfrak{b}}$ such that the the probability of transitioning from state $S_{\mathfrak{a}}$ to $S_{\mathfrak{b}}$ is proportional to $r_{\mathfrak{a}\mathfrak{b}}$.

2. Increment the current time by drawing from the distribution in Eq. (1). This can be achieved by choosing a uniform random number $\xi \in (0, 1]$ and setting the time increment $\Delta t = -Q_{\mathfrak{a}}^{-1} \log(\xi)$.

Note that while the method is written down for a finite number $\mathfrak{N}$ of states in Algorithm 1, it also works for systems with an infinite number of configurations, if it is assumed that in each step of the algorithm only a finite number of other states can be reached. This is often a sensible assumption since particles can only hop to nearby sites.

For a particular problem the propensities $r_{\mathfrak{a}\mathfrak{b}}$ are an input for the algorithm and need to be calculated, for example by working out the Boltzmann factors of different configurations. Crucially, this calculation requires knowledge of the change $\Delta U_{\mathfrak{a}\mathfrak{b}} = U_{\mathfrak{b}} - U_{\mathfrak{a}}$ in system energy induced by the hop. Including the contribution of the electrostatic interaction to this energy difference is very expensive and requires efficient algorithms.

2.2. The Fast Multipole Method

To allow an in-depth understanding of the proposed new algorithm for electrostatic interactions in KMC, we first describe the classical Fast Multipole Method (FMM) introduced in [16], before discussing its adaptation in Section 3. For further technical details we refer the reader to the original literature [16, 17, 18]. In

Algorithm 1 Kinetic Monte Carlo (KMC) method for generating snapshots $S^{(0)}, S^{(1)}, \ldots, S^{(n)}$ of the system dynamics.

Pick initial state $S^{(0)}$, set $t = 0$, $n = 0$
while $t < T$ **do**
  Set i such that $S^{(n)} = S_{\mathfrak{a}}$
  for all states $\mathfrak{b} = 1, \ldots, \mathfrak{N}$ **do**
    Calculate difference $\Delta U_{\mathfrak{a}\mathfrak{b}} = U_{\mathfrak{b}} - U_{\mathfrak{a}}$
    Derive propensities $r_{\mathfrak{a}\mathfrak{b}} = r(\Delta U_{\mathfrak{a}\mathfrak{b}})$
    Calculate $R_{\mathfrak{a}\mathfrak{b}} = \sum_{\mathfrak{c}=1}^{\mathfrak{b}} r_{\mathfrak{a}\mathfrak{c}}$, $Q_{\mathfrak{a}} = R_{\mathfrak{a}\mathfrak{N}}$
  end for
  Draw a uniform random number $\zeta \in (0, 1]$
  Set $S^{(n+1)} = S_{\mathfrak{b}}$ with $R_{\mathfrak{a},\mathfrak{b}-1} < Q_{\mathfrak{a}}\zeta \leq R_{\mathfrak{a}\mathfrak{b}}$
  Draw another uniform number $\xi \in (0, 1]$
  Calculate time increment $\Delta t = -Q_{\mathfrak{a}}^{-1} \log(\xi)$
  Set $n \mapsto n + 1$, $t \mapsto t + \Delta t$
end while

three dimensions the FMM algorithm uses a hierarchical grid with L levels for the computational domain Ω (which is assumed to be a equilateral 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 typically L is chosen such that there 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, followed by an upward- and downward traversal of the grid hierarchy (see Fig. 2). In the upward pass of the algorithm the multipole expansions around the centre of a cell are recursively combined and converted to multipole expansions around the centre of the parent cell, obtaining a single multipole expansion around the centre of the computational domain on the coarsest level $\ell = 1$. 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 contribution from distant charges are resolved at the appropriate level of accuracy, while including the contribution from closer charges in finer levels. The p-term multipole expansions Φ and the local expansions Ψ which play a central role in the FMM algorithm can be expressed in terms of the spherical harmonics $Y_n^m(\theta, \phi)$, i.e.

$$\Phi(r, \theta, \phi) = \sum_{n=0}^{p} \sum_{m=-n}^{+n} M_n^m r^{-(n+1)} Y_n^m(\theta, \phi), \qquad \Psi(r, \theta, \phi) = \sum_{n=0}^{p} \sum_{m=-n}^{+n} L_n^m r^n Y_n^m(\theta, \phi). \tag{2}$$

where (r, θ, ϕ) are spherical coordinates relative to a suitable origin.

Overall it can be shown [16, 17, 18] that at leading order the computational cost of the method is

$$\text{Cost}_{\text{FMM}} = Cp^4 N + \ldots$$

where p is the order where the multipole and local expansions in Eq. (2) are truncated. To bound the error by some ϵ, the value of p needs to be chosen such that $p = \mathcal{O}(\log_2 \epsilon)$.

The calculation of the local expansions on the finest level is written down explicitly in Algorithm 2 in Appendix A and requires the following definitions, which will be used in the subsequent discussion of the FMM method for KMC simulations:

$\Phi_{\ell,\alpha}$ the p-term multipole expansion (see Eq. (2)) about the centre of cell α on level ℓ that describes the potential induced by all charges contained in cell α.

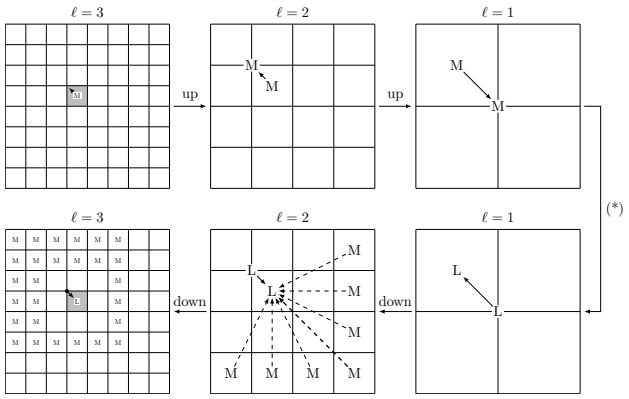

Figure 2: Overview of the FMM field, see Algorithm 2 in Appendix A for details. The translation of multipole expansions $\Phi_{\ell,\alpha}$ in the upward pass is shown in the first row. The second row shows the conversion of multipole expansions in the interaction list into local expansions $\Psi_{\ell,\alpha}$, which are represented on the next finer level. The asterisk on the right hand side of the figure stands for any additional operations on the coarsest level to account for the boundary conditions (see Section 2.2.1).

$\Psi_{\ell,\alpha}$ the p-term local expansion (see Eq. (2)) about the centre of cell α on level ℓ that describes the potential induced by all charges outside the cell α and its 26 nearest neighbours.

$\mathcal{T}_{\mathrm{MM}}$ the linear operator translating multipole moments to multipole moments around a different origin.

$\mathcal{T}_{\mathrm{ML}}$ the linear operator converting multipole moments to coefficients of the local expansion.

$\mathcal{T}_{\mathrm{LL}}$ the linear operator translating local expansion coefficients to local expansion coefficients around a different origin.

Finally, the short range contribution of the electrostatic potential is obtained by calculating the field generated by charges in neighbouring cells directly. Fig. 3 illustrates how the total calculation is split up into the long- and short-range contributions discussed above. The algorithm for calculating the total electrostatic energy from the local expansions $\Psi_{L,\alpha}$ on the finest level is given in Algorithm 3 in Appendix A.

2.2.1. Boundary conditions

So far we assumed free-space boundary conditions, i.e. we consider a finite charge distribution contained inside an unbounded physical domain. In this case the interaction list is empty on the two coarsest levels, and we can set $\Psi_{1,1} = \overline{\Psi}_{1,1} = \Psi_{2,\alpha} = \overline{\Psi}_{2,\alpha} = 0$ or equivalently skip those two levels in the downward pass.

When simulating large physical systems, however, the computational domain is typically replicated in one or several space dimensions to avoid spurious surface effects from the finite computational domain. The FMM algorithm is readily modified to account for this, as we discuss in the following for two important cases. Note that care has to be taken if the system has a net charge or a non-zero dipole moment - in those cases the lowest-order sums over periodic copies is conditionally convergent and need to be fixed using physical conditions, see appendix 4.1 of [16] and Appendix B of this paper.

Periodic. In the simplest case the computational domain is replicated periodically. To account for this, the operator $\mathcal{T}_{\mathrm{ML}}$ has to be modified on the coarsest level. On this level the local expansion receives contributions from the multipole expansions in an infinite number of well-separated periodic copies. In [29] the contributions from those copies are summed with an Ewald-like method and it is shown that they can be accounted for by simply replacing the spherical harmonics Y_ℓ^m which appear in the linear operator $\mathcal{T}_{\mathrm{ML}}$ by an infinite sum R_ℓ^m. In other words, the local expansion $\Psi_{1,1}$ on the coarsest level can be obtained from the multipole expansion $\Phi_{1,1}$ on the same level through multiplication by a known linear operator: $\Psi_{1,1} = \mathcal{R}\Phi_{1,1}$. The sum R_ℓ^m and the linear operator $\mathcal{R}$ can be calculated once at the beginning of the simulation.

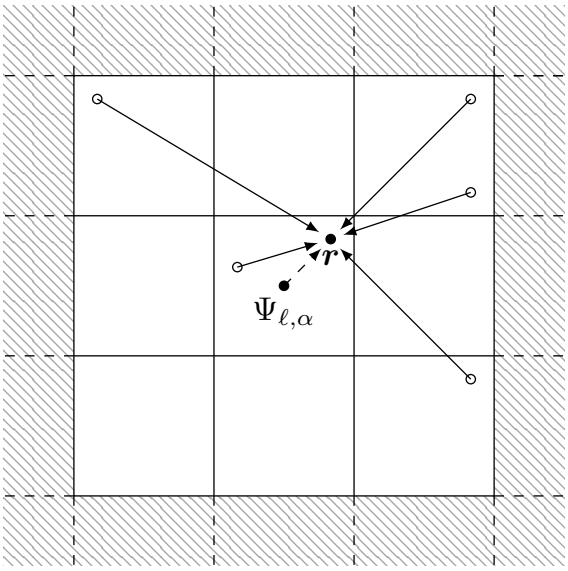

Figure 3: The potential field at $\boldsymbol{r}$ is given by the sum of (1) the evaluation of $\Psi_{\ell,\alpha}$ at $\boldsymbol{r}$ (long range) and (2) direct interactions with charges in the white region (short range). Each solid arrow illustrates a direct interaction with a charge represented by an empty circle.

Dirichlet. In some cases it is desirable to apply homogeneous Dirichlet boundary conditions $\phi(\boldsymbol{x}) = 0$ on some or all boundaries of the computational domain. This allows the inclusion of an external electric field which is present for example in batteries. As discussed in Appendix 4.2 of [16], this can be achieved by adding appropriate virtual mirror charges, effectively replacing the infinite grid of periodic copies by suitably modified reflected and inverted copies of the primary charge distribution. Apart from adjusting the value of the sum R_ℓ^m this will require no further modifications of the algorithm. An alternative approach, which we pursue here (and which is also described in [20]), is to simply extend the computational domain with the first mirror charge image and replicating the extended domain periodically, as shown in Fig. 4. We then apply the above algorithm for periodic boundary conditions to this extended domain. In the physical applications we consider here the potential is fixed at the top and bottom of the domain, which is assumed to be periodic in the other two space dimensions. This will only lead to a potential loss of performance by at most a factor two from doubling the number of charges in the system.

2.2.2. Dipole correction

In the case of non-trivial boundary conditions, the contribution of the dipole terms to the infinite sum which determines the local expansion is conditionally convergent. As discussed in [16], the value of the sum needs to be fixed by physical considerations. For example, following section 4.1 of [16], one could require that for a configuration which consists of a pure dipole pointing in the z direction the difference $\Delta\phi = \phi(\boldsymbol{r}_{a/2}) - \phi(-\boldsymbol{r}_{a/2})$ in the potential between the points $\boldsymbol{r}_{a/2} = (0, 0, a/2)$ and $-\boldsymbol{r}_{a/2}$ vanishes. This is not the case for the treatment in [29], which induces a constant electric field in the z-direction in the presence of a dipole; physically this corresponds to a non-zero surface charge at infinity. In our calculations we choose to require $\Delta\phi = 0$, i.e. no surface charge at infinity. As explained in Appendix B, this can be achieved by adding a compensating external electric field $\boldsymbol{E} = \frac{4\pi}{3}\boldsymbol{p}$ where $\boldsymbol{p}$ is the dipole moment in the simulation cell. In practice this amounts to modifying the local expansion coefficients L_1^m defined in Eq. (2) on the coarsest level.

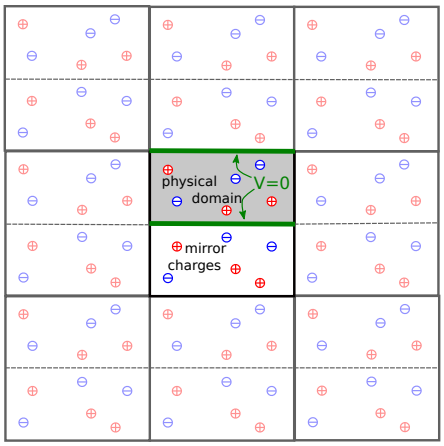

Figure 4: Computational domain used for zero-potential Dirichlet boundary conditions $V = 0$ at the top and bottom of the device, which is highlighted in green. After duplicating the charges in the physical domain (gray background) with mirror charges, the entire computational domain (black box) is replicated periodically.

3. FMM for KMC

We now describe how the FMM algorithm can be used to compute electrostatic energy differences required for the calculation of propensities in KMC simulations. For simplicity, we initially consider free-space boundary conditions, before discussing the modifications which are required to adapt the algorithm to periodic- and Dirichlet- boundary conditions in Section 3.1. As we will show, using the correctly modified KMM results in a total computational complexity of $\mathcal{O}(p^2N)$ per KMC step (the complexity is $\mathcal{O}(p^4N)$ for non-trivial boundary conditions). In each step, the electrostatic calculation can be split into two parts:

- **Propose moves**: calculate the change in electrostatic energy $\Delta U_{\mathfrak{ab}} = U_{\mathfrak{b}} - U_{\mathfrak{a}}$ for all potential moves (line 5 of Algorithm 1)
- **Accept move**: update the electrostatic potential, i.e. the local expansions $\Psi_{L,\alpha}$, once a move has been accepted (line 10 of Algorithm 1).

We further assume that the standard FMM algorithm in Algorithm 2 has been used to calculate an expansion of the local field $\Psi_{L,\alpha}$ from long-range contributions before the first KMC step. Since the number of KMC moves is very large (compared to p^2), this $\mathcal{O}(p^4N)$ startup cost can be safely neglected. In the following the calculation of electrostatic energy differences in the proposal stage and the update of FMM data structures are discussed separately.

Propose moves. Let $\boldsymbol{r}'$ be the proposed new position of a particle with charge q currently located at position $\boldsymbol{r}$ in cell α on the finest level L of the FMM grid hierarchy. Further denote the 26 direct neighbours of a cell α as $\mathcal{N}_b(\alpha)$ and define $\overline{\alpha} = \alpha \cup \mathcal{N}_b(\alpha)$. Assuming that the new position is in cell α' (which might be identical to α), the total change in the electrostatic energy due to the move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is given by

$$\Delta U_{\boldsymbol{r}'\leftarrow\boldsymbol{r}} = q\left(\Psi_{L,\alpha'}(\boldsymbol{r}') + \sum_{\boldsymbol{r}^{(i)}\in\overline{\overline{\alpha'}}} \frac{q^{(i)}}{\left|\boldsymbol{r}' - \boldsymbol{r}^{(i)}\right|}\right) - q\left(\Psi_{L,\alpha}(\boldsymbol{r}) + \sum_{\substack{\boldsymbol{r}^{(i)}\in\overline{\overline{\alpha}} \\ \boldsymbol{r}^{(i)}\neq\boldsymbol{r}}} \frac{q^{(i)}}{\left|\boldsymbol{r} - \boldsymbol{r}^{(i)}\right|}\right) - \frac{q^2}{|\boldsymbol{r}' - \boldsymbol{r}|}. \tag{3}$$

The first two brackets in Eq. (3) describe the difference in electrostatic energy of the particle at the new and old position, split into a long range part (given by the local expansions $\Psi_{L,\alpha}$ and $\Psi_{L,\alpha'}$) and direct interactions with all particles in the cell which contains the particle and its direct neighbours. Note that the

terms in the first bracket, which describes the potential energy after the move, contain the potential induced by the particle at position $\boldsymbol{r}$ before the move. This contribution is contained either implicitly in $\Psi_{L,\alpha'}$, if the cells α and α' are well separated, or included in the direct contribution if this is not the case, because one of the $\boldsymbol{r}^{(i)}$ is identical to $\boldsymbol{r}$. Clearly this is incorrect since the particle has moved to $\boldsymbol{r}'$ in this proposal and is no longer at $\boldsymbol{r}$. This is fixed by removing the spurious self-interaction in the final term of Eq. (3).

Since the local expansion $\Psi_{L,\alpha}$ defined in Eq. (2) consists of $\mathcal{O}(p^2)$ terms, and each cell contains $\overline{N}_{\text{local}} = \mathcal{O}(1)$ particles on average, the total cost per proposal is

$$\text{Cost}^{(\text{free})}_{\text{propose}} = Cp^2 + 27\overline{N}_{\text{local}} = \mathcal{O}(p^2), \tag{4}$$

independent of the total number of charges.

Accept move. Once a move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ has been accepted, the local expansion $\Psi_{L,\beta}$ on the finest level has to be updated in all cells β to account for this. Naively, this could be done by re-calculating the entire field using Algorithm 2, at a cost of $\mathcal{O}(p^4 N)$. However, since the change in the charge distribution is only very small, the change in $\Psi_{L,\beta}$ can be computed much more efficiently by subtracting the contribution of the charge at the original position $\boldsymbol{r}$ and adding it back on at the new position $\boldsymbol{r}'$. For this, loop over all cells β on the finest level. If cell β is well separated from cell α (i.e. $\beta \notin \alpha \cup \mathcal{N}_b(\alpha)$), add the local expansion around the centre of cell α which is induced by a monopole with charge $-q$ at position r to $\Psi_{L,\alpha}$, using a multipole-to-local translation $\mathcal{T}_{\text{ML}}$. Similarly, if β is well separated from α', add the local expansion induced by a monopole of charge $+q$ at the new position $\boldsymbol{r}'$. Since the local expansion contains $\mathcal{O}(p^2)$ terms and the total number of cells on the finest level is $\mathcal{O}(N)$, this reduces the cost of the accept step to

$$\text{Cost}^{(\text{free})}_{\text{accept}} = C'p^2 N = \mathcal{O}(p^2 N). \tag{5}$$

For higher degrees p this leads to significant savings of a factor p^2 relative to the naive re-calculation with Algorithm 2.

The above method is readily extended to non-trivial boundary conditions introduced in Section 2.2.1 as follows.

3.1. Boundary conditions

Periodic boundary conditions. When periodic boundary conditions are applied, the simulation cell is surrounded by an infinite lattice of periodic images of the domain Ω indexed by integer valued offsets $\boldsymbol{\nu} \in \mathbb{Z}^3$ (where $\boldsymbol{\nu} = \mathbf{0}$ corresponds to the primary image). This infinite lattice is split into two disjoint sets $\mathbb{Z}^3 = V^{(\square)} \cup V^{(\infty)}$ as shown in Fig. 5. The finite set

$$V^{(\square)} = \{\boldsymbol{\nu} : |\nu_k| \leq 1 \text{ for all } k = 1, 2, 3\}$$

contains the primary image and the surrounding 26 nearest neighbours and

$$V^{(\infty)} = \{\boldsymbol{\nu} : |\nu_k| > 1 \text{ for at least one } k = 1, 2, 3\}$$

consists of all other periodic copies. Due to linearity, the system energy is given by a contribution from periodic images in $V^{(\square)}$ plus the contribution from images in $V^{(\infty)}$. For a proposed move we consider the contribution to system energy from each of these two sets separately and sum them to obtain the total change in system energy.

We refer to the contribution to the system energy from images in $V^{(\infty)}$ as the *far-field* component and the contribution from images in $V^{(\square)}$ as the *near-field* component. As before, let $\boldsymbol{r}'$ be the proposed new position of a particle with charge q currently located at $\boldsymbol{r}$ in cell α on the finest level L of the FMM grid hierarchy and assume that the new position $\boldsymbol{r}'$ is in cell α'.

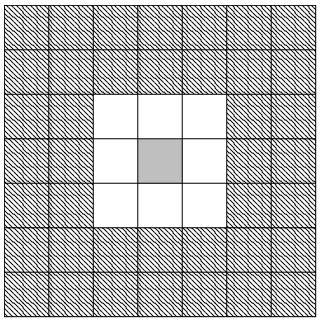

Figure 5: Sketch of $V^{(\square)}$ and $V^{(\infty)}$ defined in Section 3.1. $V^{(\infty)}$ contains the outer hatched cells and $V^{(\square)}$ consists of the primary image (solid grey) and the surrounding nearest neighbours (empty cells).

Near-field component. To compute the near-field component we simply extend the method described for free-space boundary conditions to include all images in $V^{(\square)}$; instead of moving a single particle, we also move its 26 copies in $V^{(\square)}$ when proposing a move.

While this could be achieved by working with all charges in the 27 cells in $V^{(\square)}$ and employing free-space boundary conditions, in practice it is more efficient to work with the primary image only and implicitly include the 26 copies. To achieve this, first the FMM algorithm in Algorithm 2 is modified such that at the end of the downward pass a local expansion $\Psi_{L,\alpha}$ represents the potential induced by all charges in $V^{(\square)} \setminus \mathcal{N}_b(\alpha)$. This can be realised by setting $\Psi_{1,1} = \overline{\Psi}_{1,1} = \Psi_{2,\alpha} = \overline{\Psi}_{2,\alpha} = 0$ at the beginning of the downward pass. Secondly, the interaction lists used in the downward pass are not truncated at the boundary of the primary domain and instead are wrapped around the boundary to account for the neighbours in $V^{(\square)}$. In a similar manner the neighbour cells $\mathcal{N}_b(\alpha)$ of a boundary cell α include cells over the boundary in $V^{(\square)}$. With these modifications the contribution to the system energy from charges in $V^{(\square)}$ is

$$U^{(\square)} = \frac{1}{2}\sum_{i=1}^{N} q^{(i)}\omega^{(i)} \qquad \text{with}$$

$$\omega^{(i)} = \Psi_{L,\alpha[i]}(\boldsymbol{r}^{(i)}) + \sum_{\substack{\boldsymbol{r}^{(j)} \in \overline{\overline{\alpha[i]}}^{*} \\ i \neq j}} \frac{q^{(j)}}{\left|\boldsymbol{r}^{(i)} - \boldsymbol{r}^{(j)}\right|}$$

where $\alpha[i]$ is the index of the fine-level cell which contains particle i. The asterisk $(*)$ on $\overline{\overline{\alpha[i]}}^{*}$ indicates that neighbours are wrapped periodically across the boundary of the computational domain, as discussed above. The factor $\frac{1}{2}$ is required to avoid double-counting in the total energy.

Consequently, the change in the near-field system energy for the proposed move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is

$$\begin{aligned}\Delta U^{(\square)}_{\boldsymbol{r}'\leftarrow\boldsymbol{r}} = q\left(\Psi_{L,\alpha'}(\boldsymbol{r}') + \sum_{\boldsymbol{r}^{(i)} \in \overline{\overline{\alpha'}}^{*}} \frac{q^{(i)}}{\left|\boldsymbol{r}' - \boldsymbol{r}^{(i)}\right|}\right) - q\left(\Psi_{L,\alpha}(\boldsymbol{r}) + \sum_{\substack{\boldsymbol{r}^{(i)} \in \overline{\overline{\alpha}}^{*} \\ \boldsymbol{r}^{(i)} \neq \boldsymbol{r}}} \frac{q^{(i)}}{\left|\boldsymbol{r} - \boldsymbol{r}^{(i)}\right|}\right) \\ - \; q^2 \sum_{\boldsymbol{\nu} \in V^{(\square)}} \frac{1}{|\boldsymbol{r}' - (\boldsymbol{r} + a\boldsymbol{\nu})|} + q^2 \sum_{\substack{\boldsymbol{\nu} \in V^{(\square)} \\ \boldsymbol{\nu} \neq \boldsymbol{0}}} \frac{1}{|\boldsymbol{r}' - (\boldsymbol{r}' + a\boldsymbol{\nu})|}.\end{aligned} \tag{6}$$

Eq. (6) is identical to Eq. (3) except for the final two terms. The penultimate term is a generalisation of the self-energy correction in Eq. (3), which also includes the periodic copies of $\boldsymbol{r}$. The final term accounts for the fact that all periodic copies of the particle are moved to new positions $\boldsymbol{r}' + a\boldsymbol{\nu}$ with $\boldsymbol{0} \neq \boldsymbol{\nu} \in V^{(\square)}$, and those charges contribute to the energy of the particle at $\boldsymbol{r}'$. The sum is independent of $\boldsymbol{r}'$ and it is readily

evaluated to

$$\sum_{\substack{\boldsymbol{\nu}\in V^{(\square)}\\ \boldsymbol{\nu}\neq\mathbf{0}}} \frac{1}{|\boldsymbol{r}'-(\boldsymbol{r}'+a\boldsymbol{\nu})|} = \frac{1}{a}\sum_{\substack{\boldsymbol{\nu}\in V^{(\square)}\\ \boldsymbol{\nu}\neq\mathbf{0}}} |\boldsymbol{\nu}|^{-1} = \frac{1}{a}\left(6+\frac{8}{\sqrt{3}}+\frac{12}{\sqrt{2}}\right) \approx \frac{19.104}{a}$$

Far-field component. We compute the potential induced by charges in $V^{(\infty)}$ by following the approach used for the standard FMM algorithm [29]. In the setup phase of the simulation the multipole expansion $\Phi_{1,1}$ is computed directly from the initial configuration. The $(n,m)^{\text{th}}$ multipole coefficient is

$$K_n^m = \sum_{i=1}^{N} q^{(i)} r_i^n Y_n^{-m}(\theta_i,\phi_i).$$

This is converted into the local expansion $\Psi_{1,1}$ with expansion coefficients $H_n^m = \mathcal{R}\left(K_n^m\right)$ where $\mathcal{R}$ is the operator introduced in Section 2.2.1. More specifically, the local expansion of the potential induced by the periodic images in $V^{(\infty)}$ is (c.f. Eq. (2))

$$\varphi(r,\theta,\phi) = \sum_{n=0}^{p}\sum_{m=-n}^{n} H_n^m r^n Y_n^m(\theta,\phi). \tag{7}$$

Hence the energy of N charges interacting with a far-field φ which is expressed as the local multipole expansion in Eq. (7) is obtained by evaluating Eq. (7) at the particle positions $\boldsymbol{r}^{(i)} = (r_i,\theta_i,\phi_i)$, multiplying by $q^{(i)}$ and summing over all particles.

$$U^{(\infty)} = \sum_{i=1}^{N} q^{(i)} \sum_{n=0}^{p}\sum_{m=-n}^{n} H_n^m r_i^n Y_n^m(\theta_i,\phi_i) \tag{8}$$

$$= \sum_{n=0}^{p}\sum_{m=-n}^{n} E_n^m H_n^m, \quad \text{where} \quad E_n^m = \sum_{i=1}^{N} q^{(i)} r_i^n Y_n^m(\theta_i,\phi_i). \tag{9}$$

The double sum in Eq. (9) is readily computed as a dot product in $\frac{1}{2}(p+1)(p+2)$ dimensions at a cost of $\mathcal{O}(p^2)$. Now consider a proposed move of a particle with charge q from $\boldsymbol{r}$ to $\boldsymbol{r}'$. To evaluate the far-field energy of the modified charge distribution we need to do two things: first, we need to add $qr'^n Y_n^m(\theta',\phi')$ to the expression for E_n^m in Eq. (9) and simultaneously subtract $qr^n Y_n^m(\theta,\phi)$ since the position of the particle has changed in the sum in Eq. (8). Secondly, the far-field multipole coefficients K_n^m and hence the local expansion coefficients H_n^m in Eq. (7) change, since we assume that this far-field is generated by the periodic copies of the cell in $V^{(\infty)}$. This can be accounted for by adding a correction to the multipole expansion K_n^m which describes a monopole of charge $+q$ at $\boldsymbol{r}'$ and a monopole of charge $-q$ at the old position $\boldsymbol{r}$. Those two modifications are achieved by setting

$$\bar{E}_n^m = E_n^m + q\left(r'^n Y_n^m(\theta',\phi') - r^n Y_n^m(\theta,\phi)\right) \tag{10}$$

and

$$\bar{K}_n^m = K_n^m + q\left(r'^n Y_n^{-m}(\theta',\phi') - qr^n Y_n^{-m}(\theta,\phi)\right). \tag{11}$$

This gives the modified local expansion coefficients

$$\bar{H}_n^m = \mathcal{R}\left(\bar{K}_n^m\right).$$

The new far-field contribution to the system energy for the proposed move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is

$$U_{\boldsymbol{r}'}^{(\infty)} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \bar{E}_n^m \bar{H}_n^m,$$

Total change in energy. Combining the near-field and far-field components the change in system energy of a proposed move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is given by

$$\Delta U_{\boldsymbol{r}' \leftarrow \boldsymbol{r}} = \Delta U^{(\square)}_{\boldsymbol{r}' \leftarrow \boldsymbol{r}} + U^{(\infty)}_{\boldsymbol{r}'} - U^{(\infty)}_{\boldsymbol{r}}.$$

The near-field component retains the $\mathcal{O}(p^2)$ computational complexity of the free-space case as the additional correction terms have a constant cost per proposal. For the far-field energy evaluation, the creation of $\bar{E}$ and $\bar{H}$ involves the computation of $4p^2$ coefficients with complexity $\mathcal{O}(p^2)$. The multipole-to-local operator $\mathcal{R}$ can be applied as a matrix-vector product with computational complexity $\mathcal{O}(p^4)$. Finally, evaluation of the far-field local expansion exhibits a $\mathcal{O}(p^2)$ computational complexity. Hence the overall cost of proposing a move is

$$\text{Cost}^{(\text{BC})}_{\text{propose}} = \mathcal{O}(p^4), \tag{12}$$

independent of the number of charges. This should be compared to the $\mathcal{O}(p^2)$ complexity for free-space boundary conditions given in Eq. (4).

Accept move. For an accepted move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ the local expansions on the finest level $\Psi_{L,\beta}$ are updated as in the free-space case. This operation has a computational complexity of $\mathcal{O}(p^2 N)$. To account for the far-field component, we store the quantities E^m_n and H^m_n. Whenever a move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is accepted, the values are updated according to Eqs. (10) and (11), i.e.

$$E^m_n \leftarrow \bar{E}^m_n, \qquad\qquad K^m_n \leftarrow \bar{K}^m_n.$$

This update requires the computation of $4p^2$ expansion terms, and hence has an $\mathcal{O}(p^2)$ cost. For an accept operation in the fully periodic case the dominant cost is the update of the local expansions $\Psi_{L,\beta}$ required for the near-field energy computation. We conclude that the total cost for accepting a move is

$$\text{Cost}^{(\text{BC})}_{\text{accept}} = \mathcal{O}(p^2 N). \tag{13}$$

which has the same computational complexity as the accept-step for free-space boundary conditions[2] in Eq. (5).

We conclude that the total complexity of the algorithm is $\mathcal{O}(p^4 N)$ per KMC step even for non-trivial boundary conditions. The estimates in Eqs. (12) and (13) are confirmed numerically in Fig. 9 below.

4. Related work

To highlight the impact of the novel FMM-based algorithm presented in this paper, we review existing approaches for including electrostatic interactions in KMC simulations.

As argued in [20], most recent KMC studies truncate or neglect long range electrostatic interactions. The few exceptions reported in the literature typically include some results obtained with the Ewald method, which serves as a computationally expensive reference implementation to quantify systematic errors introduced by those approximations. For example, the authors of [21] do not include the effect of electrostatic interactions in most of their results for a KMC study in doped perovskites. This leads to a change in the protonic diffusion coefficient by 14%, compared to the "correct" result obtained with Ewald summation. A systematic comparison of photovoltaic simulations with truncated potentials and the Ewald method is also presented in [20]. The authors find that introducing a cutoff potential underestimates the device performance and overestimates average charge carrier densities. The Ewald summation is optimised by precomputing the mutual potential between all pairs of charges. This reduces the cost of one proposal to $\mathcal{O}(N)$. On the structured lattice which is used in [20] it requires the storage of $\mathcal{O}(N)$ terms, but for large systems with

[2] To keep the interface in the code general and allow transitions to states which have not been previously proposed, the change in energy for the new state will be re-computed when accepting a move. This adds an additional $\mathcal{O}(p^4)$ cost, which can be safely neglected as long as $p^2 \ll N$.

an irregular arrangement of the hopping sites the $\mathcal{O}(N^2)$ memory requirements would make the method infeasible. In [30] perovskite crystal growth is modelled with a KMC method which uses Ewald summation for long range electrostatic interactions. However, the setup is very different to the problem considered here, since the system is described by a series of growing stacks on a 2d surface.

Electrostatic interactions can also be included by mapping the charge distribution to a grid and solving the Poisson equation. Since a solve of this three-dimensional partial differential equation is expensive, a lower dimensional approximation is used in some cases to make the computation feasible. For example, in [23, 24] the one dimensional Poisson equation is solved for a layer averaged charge density, while including the electrostatic field of nearby charges and periodic mirror images exactly. Naturally, this approach introduces uncontrollable systematic errors.

As discussed in [22], including electrostatics by solving the three dimensional Poisson equation naively requires an expensive re-calculation of the potential for every potential hop since the charge has moved and its contribution can not be included in the current potential. To address this issue, the self-interaction error in the naive approach can be suppressed or removed by adding and subtracting the field of a single charge. While efficient methods (such as multigrid [31]) exist for solving the Poisson equation in $\mathcal{O}(n)$ time on a grid with n cells (and it is reasonable to assume that the number of grid cells is at the same order as the number of lattice sites), discretising the Poisson equation is likely to lead to uncontrollable errors due to the peaked distribution of point particles, which can not be represented accurately on a grid. A massively parallel GPU implementation of a KMC method is described in [32]. While the authors still employ a cutoff for the electrostatic interactions, after each KMC step the local potential is updated by adding a dipole correction term, instead of recalculating the value of the field.

Existing FMM implementations and KMC libraries. Not surprisingly, there is a plethora of existing FMM implementations for the standard method written down in Algorithms 2 and 3. Some of those, such as ScalFMM [33] and ExaFMM [34] are specifically designed for performance and massively parallel scalability; ExaFMM also targets GPU systems [35]. Similarly, a parallel FMM implementation for heterogeneous systems is described in [36]. Other actively developed parallel libraries are FMMlib3d [37], RECFMM [38] and DashMM [39]. Often those libraries can treat more general potentials, for example, ScalFMM is kernel-free and allows the user to implement their own interaction potentials. Since ScalFMM has been highly optimised for modern multicore processors, we quantify the absolute performance of our standard FMM implementation by comparing it to ScalFMM below. For a comparison of different established FMM codes see also [40]. However, as far as we are aware, none of the above FMM libraries have been used in KMC simulations. Conversely, existing KMC libraries such as DL_AKMC [41], SPPARKS [42] and KMCLib [43] do not include support for long range electrostatic interactions or rely on one of the approximations described above (see e.g. the study in [44], which uses the commercial Bumblebee library [45]).

5. Implementation and user interface

Algorithmically optimal methods such as the FMM-KMC scheme introduced in Section 3 have to be implemented efficiently on massively parallel hardware. With the recent diversification of the hardware landscape, it is equally important that the code has a simple, intuitive user interface and runs on different chip architectures, such as manycore CPUs and multithreaded GPUs. To ensure the reproducibility of our results and allow others to benefit from the library, we now describe the design of the code in some detail.

5.1. Performance portable framework

The FMM-KMC algorithms introduced in this paper were implemented on top of the performance portable framework described in [25], which we refer to as "PPMD" in the following. The PPMD code is freely available at

`https://github.com/ppmd/ppmd`.

Our FMM-KMC implementation is provided in the `coulomb_kmc` Python package and can be downloaded from

https://github.com/ppmd/coulomb_kmc.

A recent snapshot of the code, which can be used to reproduce the results in the paper, is also provided as [46].

The overall design principle of PPMD is to provide a high-level Python user interface which is flexible enough to express fundamental looping mechanisms for interacting particles, while automatically generating highly efficient code on different hardware platforms. As discussed in Appendix C.1, using this Python interface also allows the easy and efficient implementation of the user-specific handling of the KMC data structures, such as masking forbidden moves based on the problem-dependent matrix of hopping sites.

Efficiency is achieved by using code generation techniques; once the user has expressed the fundamental interaction kernel as a short snippet of C-code, dedicated looping code which executes the entire particle- or particle-pair loop over all particles is generated. Depending in the platform, this might be realised with MPI, OpenMP or with CUDA on GPUs. High-level operations, such as the main time stepping loop, are implemented in Python and orchestrate the calls to the computationally expensive particle- or particle-pair loops. As shown in [25], the performance of the PPMD code is on a par with monolithic MD codes written in C and Fortran, such as LAMMPS [47] and DL_POLY [48, 49].

While the main application of PPMD is molecular dynamics, the PPMD interface and data structures are abstract enough to also implement KMC algorithms which are the topic of this paper. As described in [26], PPMD also includes a library for calculating electrostatic interactions via Ewald summation; this is very useful for testing the FMM algorithms developed in this paper.

Fundamental data structures and interfaces. The most important data structure in PPMD is the `ParticleDat` class. This is a distributed storage space for data associated with individual particles in the simulation. Example properties which can be stored as `ParticleDat`s are the positions, velocities and charges of all particles in the system. However, in principle any property, such as the number of local neighbours of a particle, could be stored in a `ParticleDat`. Under the hood, `ParticleDat`s are realised as wrappers to numpy arrays. In addition, global properties shared by all particles, such as the total energy of the system, can be stored in `GlobalArray` or `ScalarArray` objects, where the latter is read-only when accessed from inside a kernel.

To manipulate data, the user writes a short kernel in C which is executed over all particles or all pairs of particles. A particle-pair loop is specified by this C-kernel and a list of all `ParticleDat`s, `ScalarArray`s and `GlobalArray`s which are modified by the kernel. Access descriptors specify whether the data is read or written. Based on this specification of the particle-loop, the code generation system automatically generates efficient code for executing the kernel over all pairs of particles. Depending on the access descriptors, it also performs suitable communication calls to synchronise parallel data access and avoids write conflicts in threaded implementations.

To illustrate the idea, we show a short Python code snippet for naively calculating the total potential energy U for particles interacting via a Coulomb potential

$$U = \frac{1}{2} \sum_{\substack{\text{all pairs} \\ (i,j)}} \frac{q^{(i)} q^{(j)}}{\left| \boldsymbol{r}^{(i)} - \boldsymbol{r}^{(j)} \right|} \tag{14}$$

in code listing 1. Note that the key operation

$$U \leftarrow U + 0.5 \cdot q^{(i)} q^{(j)} / \left| \boldsymbol{r}^{(i)} - \boldsymbol{r}^{(j)} \right|$$

which is executed for all particle pairs is encoded in the C-code stored in the string `kernel_code`. Positions and charges are stored in the `ParticleDat`s `r` and `q`, while the total energy is stored in the `GlobalArray` `U`. Interested readers are referred to [25] for more details on PPMD.

5.2. FMM-KMC user interface

To implement the FMM algorithm for KMC simulations described in Section 3, we introduce a new `KMCFMM` Python class in the `coulomb_kmc` Python package. This class provides three key operations:

Listing 1: Python code for calculating the sum in Eq. (14) over all particle pairs.

```
# number of particles
npart=1000

# Define Particle Dats
r = ParticleDat(npart=npart, ncomp=3, dtype=c_double)
q = ParticleDat(ncomp=1, npart=npart, dtype=c_double)
U = GlobalArray(ncomp=1, initial_value=0.0, dtype=c_double)

kernel_code='''
  double dr_sq = 0.0;
  for (int k=0;k<3;++k) {
    double dr = r.i[k]-r.j[k];
    dr_sq += dr*dr;
  }
  U += 0.5*q.i[0]*q.j[0]/sqrt(dr_sq);
'''

# Define kernel
kernel = Kernel('naive_coulomb',
                kernel_code)

# Define and execute pair loop
pair_loop = PairLoop(kernel=kernel,
                     {'r':r(access.READ), 'q':q(access.READ), 'U':U(access.INC)})
pair_loop.execute()
```

1. **Initialisation** of the FMM fields and calculation of the system's full electrostatic energy at the start of the simulation
2. Evaluation of the electrostatic energy difference $\Delta U_{\boldsymbol{r}' \leftarrow \boldsymbol{r}}$ for all **proposals** $\boldsymbol{r}' \leftarrow \boldsymbol{r}$
3. **Acceptance** of a move selected by the user

Note that the rest of the KMC algorithm, such as the selection of a move for acceptance based on the propensities and working out the set of allowed potential moves is still the responsibility of the user. Section Appendix C.1 discusses an example of how the latter can be realised efficiently with the PPMD data structures and parallel loops. Relegating high-level control over the algorithm to the user also allows the easy extension of the basic KMC method to more advanced techniques, such as multilevel methods, or the inclusion of post-processing steps to extract physically meaningful information.

The constructor of the `KMCFMM` class is passed the initial positions and charges of all particles, stored in `ParticleDat` objects. It allows the user to choose the parameters of the FMM solver, such as number of FMM levels and expansion terms. The `initialise()` method computes the system's electrostatic energy for the initial positions of all charges based on the standard FMM algorithm described in Algorithms 2 and 3. The method also creates and initialises all data structures required for the subsequent proposed and accepted moves.

5.2.1. Proposing moves

The `KMCFMM` class provides two interfaces for calculating the change in system energy of proposed moves. The simple `propose()` method assumes that for a particle with index $i \in \{0, \ldots, N-1\}$ located at position $\boldsymbol{r}^{(i)}$ and carrying charge q_i, there is a finite set of n_i potential new positions $\mathcal{R}^{(i)} = \{\boldsymbol{r'}_1^{(i)}, \boldsymbol{r'}_2^{(i)}, \ldots, \boldsymbol{r'}_{n_i}^{(i)}\}$ which this particle could move to. The potential moves of all particles are stored as a list of tuples of the

Listing 2: Python code for creating a `KMCFMM` instance and proposing moves as described in Section 5.2.1.

```
import numpy as np

# Create KMCFMM instance
kmc_fmm = KMCFMM(positions=r, charges=q, r=4, l=12)

# Set up FMM data structure, calculate
# initial electrostatic energies
kmc_fmm.initialise()

# Tuple with proposed moves
P = (
      (3, np.array(((0.11, 0.13, 0.09), (0.90, 0.10, 0.08)))  # Particle 3
      ),
      (7, np.array(((0.45, 0.28, 0.89),))                     # Particle 7
      )
    )

# Calculate energy changes
U = kmc_fmm.propose(P)
```

form $\left(i, \mathcal{R}^{(i)}\right)$ for $i \in \{i_1, i_2, \dots\} \subseteq \{0, \dots, N-1\}$, which are passed to the `propose()` method in the form

$$\mathcal{P} = \left(\left(i_1, \mathcal{R}^{(i_1)}\right), \left(i_2, \mathcal{R}^{(i_2)}\right), \dots\right). \tag{15}$$

The change in total electrostatic energy for the move $\boldsymbol{r}'^{(i)}_k \leftarrow \boldsymbol{r}^{(i)}$ with $k \in \{1, \dots, n_i\}$ is denoted by $\Delta U_{\boldsymbol{r}'^{(i)}_k}$. The changes in energy for all potential moves of particle i are collected in the list

$$\mathcal{U}^{(i)} = \{\Delta U_{\boldsymbol{r}'^{(i)}_1}, \Delta U_{\boldsymbol{r}'^{(i)}_2}, \dots, \Delta U_{\boldsymbol{r}'^{(i)}_{n_i}}\}. \tag{16}$$

The `propose()` method returns a tuple of arrays of electrostatic energy differences $\left(\mathcal{U}^{(i_1)}, \mathcal{U}^{(i_2)}, \dots\right)$, where each $\mathcal{U}^{(i)}$ is of the form described in Eq. (16).

To encode three-dimensional vectors, the proposed new positions $\mathcal{R}^{(i)}$ are passed to the `propose()` method as a $n_i \times 3$ dimensional `numpy` array for each charge. The method returns a `numpy` array with the change in electrostatic energy for each proposed move. An example is shown in code listing 2. In this case particle 3 can move to two potential new positions, namely $\boldsymbol{r}'^{(3)}_1 = (0.11, 0.13, 0.09)$ and $\boldsymbol{r}'^{(3)}_2 = (0.90, 0.10, 0.08)$, whereas there is only one potential hop to the new position $\boldsymbol{r}'^{(7)}_1 = (0.45, 0.28, 0.89)$ for particle 7. For this setup the tuple $\mathcal{P}$ is given by

$$\mathcal{P} = \left(\left(3, \begin{bmatrix} 0.11 & 0.13 & 0.09 \\ 0.90 & 0.10 & 0.08 \end{bmatrix}\right), \left(7, \begin{bmatrix} 0.45 & 0.28 & 0.89 \end{bmatrix}\right)\right)$$

which should be compared to the Python code in Listing 2. After the call to `propose()`, the variable `U` will contain the corresponding energy differences as a tuple of numpy arrays in the format

$$U = \left(\left[\Delta U_{\boldsymbol{r}'^{(3)}_1}, \Delta U_{\boldsymbol{r}'^{(3)}_2}\right], \left[\Delta U_{\boldsymbol{r}'^{(7)}_1}\right]\right).$$

While the interface for proposing moves described here is intuitive, it is not optimal in terms of efficiency. The improved, but more sophisticated, `propose_with_dats()` interface is described in Appendix C, where we also illustrate its use for a practically relevant example.

Listing 3: Python code for accepting a move as described in Section 5.2.2.

```
p = kmc_fmm.accept((6, np.array((0.11,0.56,0.39))))
```

5.2.2. Accepting moves

The `KMCFMM` instance provides a `accept` method that accepts a proposed move, updates the system energy and updates internal data structures. To accept a move, the `accept()` is called with a tuple $(i, \boldsymbol{r}')$ consisting of a charge index i and a new position $\boldsymbol{r}'$. An example is given in code listing 3, which assumes that particle $i = 6$ moves to the new position $\boldsymbol{r}' = (0.11, 0.56, 0.39)$.

5.3. Parallelisation and optimisation

On distributed memory machines domain-decomposition is used to parallelise the KMC-FMM algorithm described in Section 3. For this the computational domain Ω is divided between MPI ranks such that each rank "owns" the charges in its local subdomain Ω_{local}. Proposals for particles owned by different MPI ranks are handled concurrently. Ω_{local} is augmented by a halo region to obtain an extended local domain $\overline{\Omega}_{\text{local}}$. It is assumed that the particles are evenly distributed and hops are limited by some maximal distance which determines the size of this halo and hence $\overline{\Omega}_{\text{local}}$. Under those conditions, which seem reasonable for many physical systems, domain decomposition will result in good load balancing and – as will be described in the following – requires very little parallel communication. The results in Section 6.3 confirm the excellent parallel scaling on up to 128 nodes.

To store local- and multipole expansions $\Psi_{\ell,\alpha}$ and $\Phi_{\ell,\alpha}$ on all levels of the grid hierarchy, a distributed "Octal Tree" (OT) data structure is set up at the beginning of the simulation. As described in Section 5.3 of [50], data can be attached to the OT using different parallel access modes. This allows suitable halo-exchanges between neighbouring MPI ranks and ensures that all children of a particular coarse level cell are owned by the same rank during the upward- and downward pass of Algorithm 2. The local expansions $\Psi_{L,\alpha}$ are calculated for all cells α on the finest level at the beginning of the simulation. The OT cells on the finest level are not necessarily aligned with the local subdomains Ω_{local}. However, each MPI rank keeps copies of the local expansions $\Psi_{L,\alpha}$ for all fine level OT cells which cover its extended domain $\overline{\Omega}_{\text{local}}$. It also maintains copies of all particle positions and charges in those cells.

When a potential move $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ is proposed, calculating the change in energy $\Delta U_{\boldsymbol{r}' \leftarrow \boldsymbol{r}}$ in Eq. (3) for free-field boundary conditions requires the evaluation of the local expansion on the finest level $\Psi_{L,\alpha}$ at the old and new positions, and knowledge of particle positions in neighbouring cells. This data is available in local copies, provided the halo is chosen large enough. Upon accepting a move, ownership is transferred if the moved particle crosses a subdomain boundary. The local expansions $\Psi_{L,\alpha}$ are updated for all cells α which are affected by this move. Note that neither evaluating the energy change for the proposals nor the update of $\Psi_{L,\alpha}$ at the end of a KMC step requires any parallel communication apart from sharing the details of the accepted move between all processors.

For non-trivial boundary conditions the *near-field* change in electrostatic energy $\Delta U^{(\square)}_{\boldsymbol{r}' \leftarrow \boldsymbol{r}}$ given by Eq. (6) can be handled in the same way as just described for free-field boundary conditions, both when proposing and accepting a move. Calculating the change in the *far-field* contribution $U^{(\infty)}$ given by Eq. (8) requires an update to the expansion coefficients H_n^m on the coarsest level and E_n^m defined in Eq. (9). Identical copies of both H_n^m and E_n^m are stored on all MPI ranks. When a move is accepted, they are updated locally on each MPI rank by removing the contribution from the particle at $\boldsymbol{r}$ and adding the contribution from the new position $\boldsymbol{r}'$.

We use OpenMP as a shared memory programming method to distribute the proposed moves on each MPI rank over available cores. This reduces load imbalances due to variations in the number of proposed moves per particle. Furthermore, in this hybrid MPI+OpenMP approach the volumes the subdomains handled by individual MPI ranks are larger than in an MPI only execution. In addition to improved load-balancing,

machine	chip	sockets	cores per socket	cores per node	MPI ranks	OpenMP threads per MPI rank
Balena	Intel Ivy Bridge E5-2650v2	2	8	16	4	4
	Intel Skylake Gold 6126	2	12	24	4	6
Isambard	Cavium ThunderX2	2	32	64	8	8

Table 1: Node configuration and process layout used for performance measurements on the Balena and Isambard machines.

this increases the ratio of subdomain volume to halo region volume, which reduces the computational work of accepting a move as fewer expansions and particles must be updated.

All performance critical operations which manipulate multipole and local expansions are implemented as auto-generated C code. This allows fixing the number of expansion coefficients p at compile-time, which enables important optimisations such as loop-unrolling and auto-vectorisation. In particular, evaluation of the spherical harmonics $Y_\ell^m(\theta, \phi)$ at a particular position $\boldsymbol{r} = (r, \theta, \phi)$ with recurrence relations (see e.g. [51, 52]) is carried out with a two-level loop nest. The bounds of the inner loop with index k depend on the current iteration of the outer loop over $\ell \in 0, \ldots, p$ which makes it virtually impossible to vectorise the code if the outer loop bound p is only fixed at runtime. However, if p is known at compile time, the loop can be unrolled to generate a long sequential list of updates, which contain the same algebraic operations and can are readily vectorised. In addition, when generating this code combinatorial factors which depend on the loop indices k and ℓ can be pre-computed at compile time. This reduces the number of floating point operations and further improves performance.

Finally, note that any further book-keeping operations required in a KMC step, such as those described in Algorithm 4, are implemented as `ParticleLoop` and `PairLoop` constructs in PPMD. They are therefore automatically parallelised on any parallel architecture, as described in detail in [50, 25].

6. Results

The KMC simulations reported in this section were carried out on the CPU nodes of the "Balena" cluster at the University of Bath, with some additional weak- and strong- parallel scaling runs on the ARM-based "Isambard" supercomputer. On Balena, Intel Ivy Bridge and Skylake nodes with 16 and 24 cores in total were used, whereas one full Cavium ThunderX2 node on Isambard contains 64 cores. The code was run in mixed MPI+OpenMP mode, the exact node configuration and process layout can be found in Tab. 1. This setup was used for all runs, unless explicitly stated otherwise below. All autogenerated code in PPMD was compiled with version 17.1.132 of the Intel compiler on Balena, using the same version of IntelMPI for distributed memory parallelisation. On Isambard GCC version 8.2.0 was used together with CRAY MPICH 7.7.6. Raw results and code snapshots are publicly available for reproducibility purposes in the archive at [46].

6.1. Error analysis

We first demonstrate that the errors in the FMM-KMC method can be systematically quantified and decrease as the number p of multipole expansion coefficients increases. In Chapter 5 of [50] we measured the error on the *total* electrostatic system energy U for the standard FMM algorithm and studied its dependency on p. In a KMC simulation, however, the quantity of interest is the *change* ΔU in the system's electrostatic energy for each proposed move. As confirmed by the results in Tab. 3, ΔU is typically several (two or more) orders of magnitude smaller than U, which is proportional to the problem size. This places tighter bounds on the allowed numerical errors, which have to be small compared to the energy change ΔU for individual proposed moves.

To quantify the relative error on ΔU, we consider a system of constant density (0.01 charges per unit volume) and record the change in energy for $M = 10^4$ proposed moves. As in [50], the charges are initially

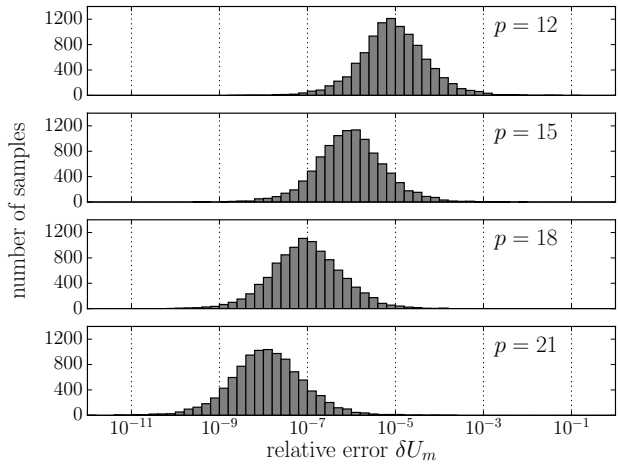

Figure 6: Histogram of the relative error δU_m defined in Eq. (17) for different numbers of expansion terms p and a system with $N = 10^4$ charges.

arranged in an almost regular cubic lattice with small random perturbations to create a setup which is representative of physically relevant systems. Each proposed move corresponds to an additional small arbitrary displacement from the initial positions. To explore a wide range of configurations, a new pseudo-random system was generated every 100 proposals.

For each proposed move m we calculate a highly accurate estimate of the "true" energy difference ΔU_m^* by using our KMC-FMM algorithm with 26 expansion terms. The corresponding energy difference computed with $p < 26$ expansion terms is denoted as ΔU_m. Based on this we define the absolute relative error of move m as

$$\delta U_m = \frac{|\Delta U_m - \Delta U_m^*|}{|\Delta U_m^*|} \tag{17}$$

and use this quantity to assess the accuracy of the simulation.

We initially keep the system size fixed at $N = 10^4$ and vary the number of expansion terms p from 12 to 21; the histograms in Fig. 6 show the number of samples with a given error for increasing p. Inspecting the distribution plotted there with logarithmic scale on the horizontal axis, it can be seen that the error on ΔU is reduced by around one order of magnitude as p is incremented by 3. To further quantify the size of the error, let

$$\langle A \rangle \equiv \frac{1}{M} \sum_{m=1}^{M} A_m$$

be the sample average over M independent proposed moves for some quantity A. We estimate the expected relative error and its variance as

$$\mathbb{E}(\delta U) \approx \langle \delta U \rangle, \qquad \mathrm{Var}(\delta U) = \mathbb{E}\left((\delta U - \mathbb{E}(\delta U))^2\right) \approx \langle \delta U^2 \rangle - \langle \delta U \rangle^2 \tag{18}$$

Corresponding estimates for the average absolute energy difference $|\Delta U^*|$ and the average absolute system energy $|U|$ are[3]

$$\mathbb{E}(|\Delta U^*|) \approx \langle |\Delta U^*| \rangle, \qquad \mathbb{E}(|U|) \approx \langle |U| \rangle.$$

[3]Since the system is initialised at random, there is an equal probability of U to be positive or negative, and it is therefore natural to consider its absolute value.

p	$\langle \delta U \rangle$	$\langle \delta U^2 \rangle - \langle \delta U \rangle^2$	$\langle \lvert \Delta U^* \rvert \rangle$	$\langle \lvert U \rvert \rangle$
12	$8.93 \cdot 10^{-5}$	$2.30 \cdot 10^{-6}$	||	||
15	$8.66 \cdot 10^{-6}$	$1.14 \cdot 10^{-8}$	$1.51 \cdot 10^{-1}$	$9.13 \cdot 10^{2}$
18	$1.24 \cdot 10^{-6}$	$6.63 \cdot 10^{-10}$		
21	$1.81 \cdot 10^{-7}$	$8.82 \cdot 10^{-12}$	||	||

Table 2: Sample average and variance of the relative error δU on the energy difference ΔU, as defined in Eqs. (17) and (18). Results are shown for a varying number of expansion terms p and fixed $N = 10^4$. The last two columns also give the sample average of the "true" energy change per proposed move ΔU^* and the total electrostatic energy U of the system.

N	$\langle \delta U \rangle$	$\langle \delta U^2 \rangle - \langle \delta U \rangle^2$	$\langle \lvert \Delta U^* \rvert \rangle$	$\langle \lvert U \rvert \rangle$
10^3	$1.58 \cdot 10^{-4}$	$7.96 \cdot 10^{-5}$	$1.41 \cdot 10^{-1}$	$8.73 \cdot 10^{1}$
10^4	$8.93 \cdot 10^{-5}$	$2.30 \cdot 10^{-6}$	$1.51 \cdot 10^{-1}$	$9.13 \cdot 10^{2}$
10^5	$8.07 \cdot 10^{-5}$	$4.36 \cdot 10^{-7}$	$1.13 \cdot 10^{-1}$	$1.81 \cdot 10^{4}$

Table 3: Sample average and variance of the relative error δU on the energy difference ΔU, as defined in Eqs. (17) and (18). Results are shown for fixed $p = 12$ and different problem sizes N. The last two columns also give the sample average of the "true" energy change per proposed move ΔU^* and the total electrostatic energy U of the system.

The dependency of $\langle \delta U \rangle$ on the number of expansion terms is shown in Tab. 2, where we also we give the estimated variance, $\langle \delta U^2 \rangle - \langle \delta U \rangle^2$ and other relevant quantities for $N = 10^4$ charges and $p \in \{12, 15, 18, 21\}$ expansion terms. We repeated the same experiment for fixed $p = 12$ and varying problem sizes N, the results are shown in Tab. 3. As confirmed by those tables, on average the relative error δU defined in Eq. (17) is about three orders of magnitude smaller than the average change in energy ΔU itself. The total electrostatic energy of the system grows in proportion with the number of charges and is significantly larger than ΔU.

6.2. Computational complexity

Next we confirm that the runtime of the method grows in direct proportion to the number of charges N. We consider a system with periodic boundary conditions. Recall that theoretically proposing a single move carries a cost of $\mathcal{O}(p^4)$ and accepting a move costs $\mathcal{O}(Np^2)$, resulting in a linear growth of the computational complexity per KMC step. This is demonstrated by varying the number of charges in a system (for a fixed number $p = 12$ of expansion terms) and plotting the time t_{propose} for an individual proposal and for accepting a move (normalised to the number of charges) t_{accept}/N in Fig. 7. As this figure shows, both times are in the range of a few μs when running the code on 16 cores of an Intel Ivy Bridge node.

The sawtooth nature of the plot is an artifact of the varying number of FMM levels, which depends on the problem size N as $L = \min(3, \lfloor \log_8(N/2) \rfloor)$. The sharp drops on the right edge of the "teeth" correspond to an increase of L by one, whereas all points on the left, shallow side of the "teeth" were obtained with the same value of L, which becomes less optimal as the problem size grows. Based on those numbers, we estimate the total time per KMC step as

$$t_{\text{step}} = N \cdot \overline{n}_{\text{propose}} \cdot t_{\text{propose}} + t_{\text{accept}}. \tag{19}$$

where $\overline{n}_{\text{propose}} = 14$ is the estimated average number of proposed moves per charge and per KMC step. This value is motivated by the setup in Section 6.3, and of the same order of magnitude as observed values for the α-NPD test case in Section 6.4. The time per KMC step t_{step} is plotted Fig. 8, which confirms that our implementation indeed achieves the expected $\mathcal{O}(N)$ computational complexity.

The results imply that it is possible to simulate a system with one million charges in about 10s per KMC step when running on a single 16 core Ivy Bridge node and limiting the relative error in the energy difference per proposed move to $\sim 10^{-3}$.

To demonstrate that the computational complexity depends polynomially on the number of expansion terms p, we repeated the above experiment but fixed the number of charges at $N = 10^5$ while varying p

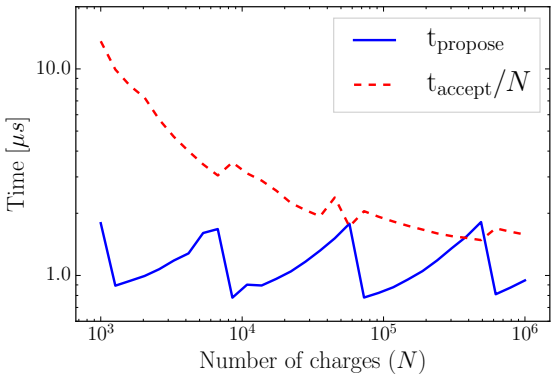

Figure 7: Time per proposed move t_{propose} and time for accepting a proposal per particle t_{accept}/N as a function of the number of charges. All results were obtained on a single Ivy Bridge node. The fluctuations in t_{accept}/N are discussed in the main text.

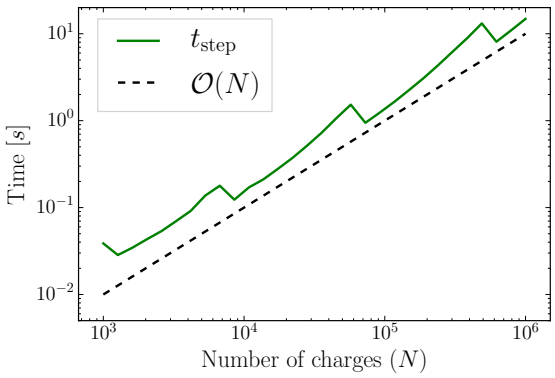

Figure 8: Total time per KMC t_{step} estimated using Eq. (19) and the numerical values for t_{propose} and t_{accept} in Fig. 7, which were obtained on a single Ivy Bridge node. The straight dashed line shows ideal linear scaling with the number of charges N.

between 2 and 30. The measured runtimes in Fig. 9 confirm that t_{accept}/N depends quadratically on the number of expansion terms, whereas t_{propose} is asymptotically proportional to the fourth power of p. For the following numerical experiments $p = 12$ expansion terms were used.

6.3. Distributed Memory Parallelism

While the previous section shows that it is possible to carry out KMC simulations with one million charged particles in a reasonable time on a single Ivy Bridge node, physically meaningful results can often only be extracted from much larger systems. Setups with $10^6 - 10^9$ particles allow the resolution of grain boundaries and ultimately move closer to the simulation of full photovoltaic devices and batteries. Modelling systems of these sizes in reasonable times requires distributed memory parallelism to utilise multiple compute nodes in a HPC facility.

As discussed in Section 5.3, our implementation employs a hybrid MPI+OpenMP parallelisation strategy. To demonstrate the parallel scaling of a full KMC simulation we implemented the book-keeping algorithm in PPMD, employing the same techniques as in the example described in Appendix C.1. Recall that this uses the `propose_with_dats()` interface of our FMM-KMC implementation for optimal efficiency. The energy differences $\Delta U_{\boldsymbol{r}' \leftarrow \boldsymbol{r}}$ of all proposed moves $\boldsymbol{r}' \leftarrow \boldsymbol{r}$ are used to calculate the associated propensities $\propto \exp\left(-\Delta U_{\boldsymbol{r}' \leftarrow \boldsymbol{r}}\right)$ by using a `ParticleLoop`. To choose a transition following steps 5-7 of the KMC Algorithm 1, the partial sums $R_{\mathfrak{ab}}$ are computed on each MPI rank and combined across all processes using an `MPI_Allgather` operation. Finally, the chosen proposed move is accepted by all MPI ranks.

To investigate how effectively our implementation scales across multiple compute nodes we performed both weak- and strong- scaling experiments. In the strong scaling experiment the problem size is kept fixed while the number of compute nodes is increased, resulting in a reduction of the total runtime. Since the local problem size decreases and hence the ratio between communication and computation usually gets worse, strong scaling is typically harder to achieve. In contrast, for the weak scaling experiment the problem size is increased in proportion to the number of nodes, in other words the *local* problem size remains constant while the total *global* problem size grows.

In both cases we consider a cubic lattice with a spacing of $h = 1.1$Å in each direction such that one in 27 sites is occupied by a charged particle. We allow proposed moves to the 14 neighbouring lattice sites which are either a distance h or $\sqrt{3}h$ away. In units of the lattice spacing the corresponding offset vectors are $(1,0,0)$, $(-1,0,0)$, $(0,1,0)$, $\ldots$, $(1,1,1)$, $(-1,1,1)$ etc. and proportional to the lattice vectors of a fcc crystalline structure. Periodic boundary conditions are applied for the electrostatic field.

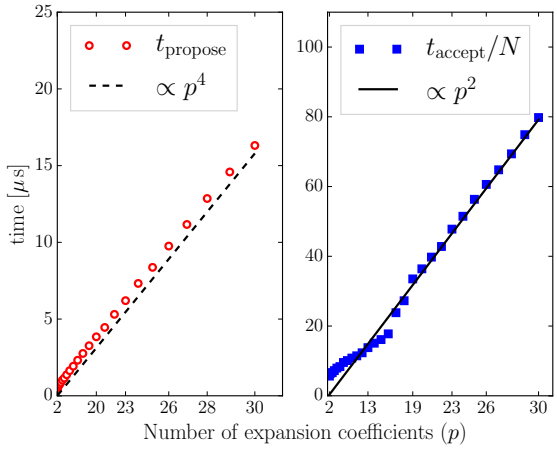

Figure 9: Time per proposal (left) and per acceptance step normalised to N (right) for varying polynomial degrees p and fixed $N = 10^5$. The horizontal axes are scaled to highlight the quartic/quadratic dependency on p. All results are obtained on an Ivy Bridge node.

For the strong scaling experiments the global problem on a single node contains 10^6 charges in total, corresponding to a cubic lattice with 300 points in each coordinate direction. On the largest node count (128 nodes on Isambard), this results in a local problem size with just under 7,800 charges per node. In the weak scaling experiment each node is responsible for 10^6 charges on average. The largest problem which was simulated contains $1.28 \cdot 10^8$ charges.

Both scaling experiments were carried out on the hardware listed in Tab. 1, comparing fully populated nodes on the different machines. For the strong scaling experiment the number of FMM levels was fixed at $L = 6$. For the weak scaling experiment the number of levels was chosen as $L = \lfloor \log_8(\gamma N) \rfloor$ where γ is tuned for each machine and depends on the relative cost of the direct interactions (lines 5-7 in Algorithm 2) and calculation of $\Psi_{L,\alpha}$ (Algorithm 3 and line 4 in Algorithm 2) on a particular hardware. On Isambard a value of $\gamma = 3.4$ was found to be optimal, whereas $\gamma = 4.7$ and $\gamma = 3.3$ turned out to give the best results on the Ivy Bridge and Skylake nodes of Balena respectively.

The time $t(P, N)$ per KMC step for this model system with $N = 10^6$ particles running on P nodes is given in Tab. 4 for different machines and also plotted in in Fig. 10 (left). As usual, the parallel efficiency $E_S(P; N)$ for the strong scaling experiment is defined relative to one node:

$$E_S(P; N) = \frac{t(1, N)}{P \cdot t(P, N)}. \tag{20}$$

For the corresponding weak scaling experiment we fix the *local* problem size, i.e. the number of charges per node, to $N_{\text{local}} = 10^6$ and increase the *total* number of charges $N = P \cdot N_{\text{local}}$ in proportion to the number of nodes. Results for $t(P, N) = t(P, P \cdot N_{\text{local}})$ are given in Tab. 5 and Fig. 10 (right), where the parallel efficiency in this weak scaling experiment is defined as

$$E(P; N_{\text{local}}) = \frac{t(1, N_{\text{local}})}{t(P, P \cdot N_{\text{local}})}. \tag{21}$$

6.4. KMC Simulation of α-NPD

We next investigate the performance of our KMC algorithm when applied to a physically realistic configuration. The system studied is α-NPD doped with F6TCNNQ, a hole transporting organic semiconductor material [53]. The scientific results of the simulations will be discussed in a forthcoming publication [54] and

P	Time per KMC step [s]					
	Isambard		Ivy Bridge		Skylake	
1	11.93	(100.0%)	21.33	(100.0%)	10.26	(100.0%)
2	5.96	(100.1%)	10.55	(101.1%)	5.09	(100.8%)
4	3.03	(98.4%)	5.36	(99.5%)	2.57	(99.7%)
8	1.48	(100.7%)	2.70	(98.7%)	1.30	(98.3%)
16	0.77	(96.3%)	1.33	(100.2%)		
32	0.42	(88.3%)	0.71	(94.2%)		
64	0.23	(79.4%)	0.39	(86.3%)		
128	0.14	(65.3%)	0.21	(79.2%)		

Table 4: Time per KMC step from a strong scaling experiment for $N = 10^6$ charges and increasing numbers of nodes P. Parallel $E_S(P; N)$ efficiency as defined by Eq. (20) is given as relative to a single node in brackets.

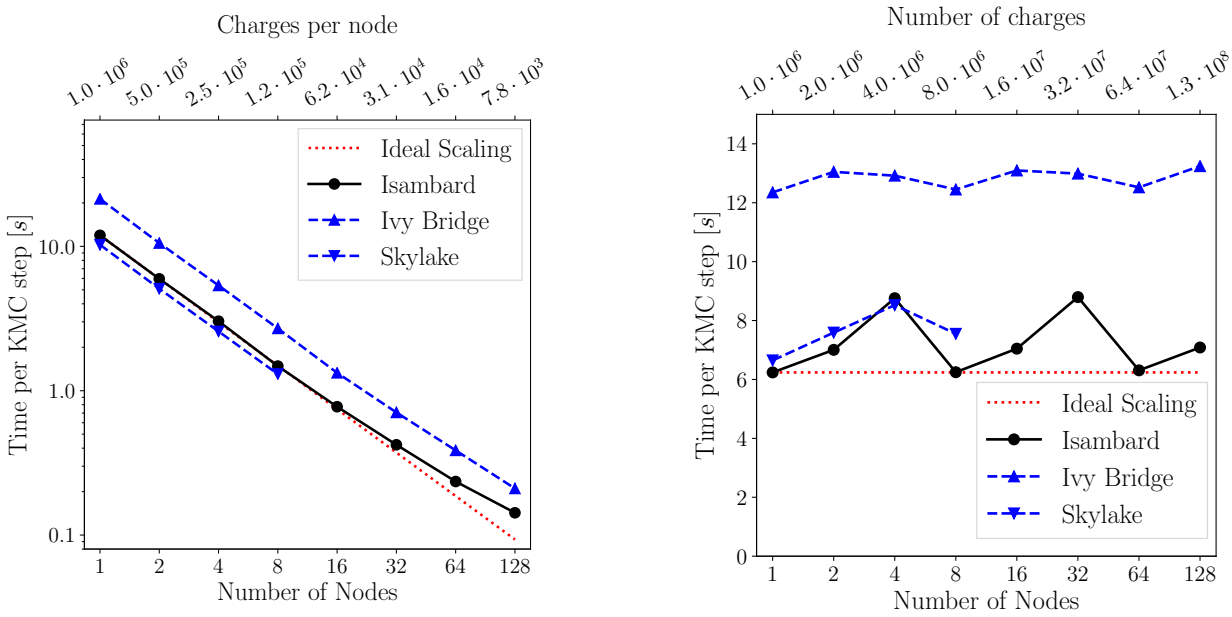

Figure 10: Strong (left) and weak (right) scaling experiments. The time per KMC step is plotted against number of compute nodes. For the strong scaling experiment the total number of charges is $N = 10^6$.

P	Time per KMC step [s]					
	Isambard		Ivy Bridge		Skylake	
1	6.24	(100.0%)	12.35	(100.0%)	6.65	(100.0%)
2	7.01	(89.1%)	13.05	(94.7%)	7.59	(87.6%)
4	8.76	(71.2%)	12.92	(95.6%)	8.52	(78.0%)
8	6.25	(99.9%)	12.45	(99.2%)	7.55	(88.1%)
16	7.05	(88.5%)	13.10	(94.3%)		
32	8.80	(70.9%)	12.99	(95.1%)		
64	6.31	(98.9%)	12.52	(98.6%)		
128	7.09	(88.1%)	13.24	(93.3%)		

Table 5: Weak scaling experiment: time per KMC step against number of nodes P with $N_{\text{local}} = 10^6$ charges per node. Parallel efficiency $E(P; N_{\text{local}})$ as defined by Eq. (21) is given as relative to a single node in brackets.

doping %	charges (N)	Time per KMC step [s] Ivy Bridge	Skylake
0.01	102	0.058	
0.05	510	0.073	
0.1	1020	0.096	
0.5	5103	0.68	0.12
1.0	10206	1.37	0.19
2.0	20412	2.74	0.35

Table 6: Time per KMC step for α-NPD simulations executed on a single node containing either two Ivy Bridge E5-2650v2 or two Skylake Gold 6126 CPUs. The Ivy Bridge simulations used 4 CPU cores and $L = 3$ FMM mesh levels. The Skylake simulations used 12 CPU cores and $L = 4$ mesh levels.

here we concentrate on evaluating the runtimes for a given setup. The dopant molecules ionise and release a mobile charge carrier; this creates a fixed negative charge and a mobile positive charge. The kinetic Monte Carlo algorithm describes the hopping of the holes between different α-NPD molecules, the hopping rates are described by Marcus theory and are functions of structural energy, temperature and molecular polarisation as well as electrostatic energy.

The KMC code used the `propose_with_dats()` interface and applied a modified version of Algorithm 4 for bookkeeping operations. This modified algorithm only updates the proposed positions $\mathcal{R}^{(i)}$ and associated masks $\mathcal{M}^{(i)}$ if charge i is in the vicinity of the previously accepted move. This removes redundant bookkeeping operations at each KMC step. The α-NPD simulations were performed with a range of applied voltages to investigate the charge mobility dependence on applied voltage. For this a constant external electric field in one direction is added to simulate a given applied potential difference across the domain. The electrostatic field induced by the charges is assumed to be periodic in all three dimensions and the charges were allowed to wrap around the simulation domain upon reaching the boundary[4].

To exploit additional parallelism between ensembles, multiple instances of the α-NPD simulation were run simultaneously. Each instance was executed on either 4 Ivy Bridge cores (running 4 simulations in parallel on a full node) or 12 Skylake cores (2 simulations per node). The dependence of the time per KMC step on the number of charges in the system is shown in Tab. 6 and plotted in Fig. 11. For the largest studied system with 20412 charges, one KMC step takes 0.35s when run on a full Skylake socket with 12 cores.

6.5. Performance comparison to ScalFMM

Finally, we verify that our implementation in the PPMD framework is indeed efficient. There is currently no existing library which implements the bespoke FMM method for KMC simulations developed in this paper. A fair comparison to other KMC packages based on Ewald summation or approximations described in [23, 24] would have to be carried out at a fixed, problem dependent error tolerance and is therefore more appropriate for future application-driven studies. A meaningful intercomparison study might also be outright impossible since most of the methods in the literature introduce uncontrollable errors. To nevertheless assess the absolute performance of our implementation we compare the runtime of the *standard* FMM method in Algorithms 2 and 3 to the freely available ScalFMM library [55]. Given that no extensive attempts have been made to optimise our code, the aim of this comparison is to verify that the performance is in the same ballpark as a published reference implementation.

For this, we measured the time spent in one iteration of a Velocity Verlet integrator for a set of particles which interact via a Coulomb potential. Although the setup of the system differed slightly (the ScalFMM test case uses free-field boundary conditions and a pure $\sim 1/r$ repulsive potential, whereas our code was

[4]Note that while this was not done here, our code also allows enforcing Dirichlet boundary conditions at the top and bottom of the domain by using mirror charges, as described in Section 2.2.1

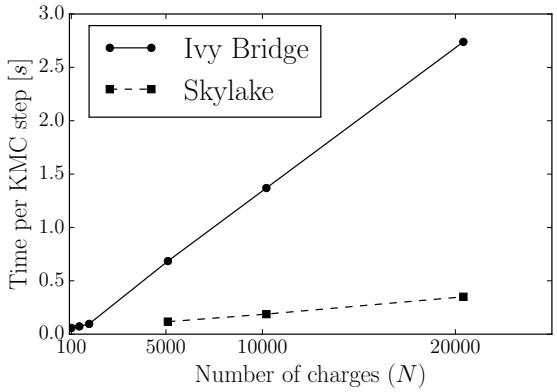

Figure 11: Time per KMC step for α-NPD simulations executed on a single node containing either two Ivy Bridge E5-2650v2 or two Skylake Gold 6126 CPUs. The Ivy Bridge simulations used 4 CPU cores and $L = 3$ FMM mesh levels. The Skylake simulations used 12 CPU cores and $L = 4$ mesh levels.

implementation	# particles	$N = 10^6$ VV-step	FMM		$N = 4 \cdot 10^6$ VV-step	FMM	
PPMD	(MPI only)	3.03	2.91	[96%]	9.78	8.96	[92%]
	(MPI+OpenMP)	3.96	3.78	[95%]	12.39	11.48	[93%]
ScalFMM	(OpenMP only)	1.16	1.14	[99%]	4.00	3.93	[98%]

Table 7: Performance comparison of our implementation of the Standard FMM algorithm to ScalFMM. The total time per Velocity Verlet (VV) step is listed in the first column and includes overheads from updating the velocities and positions with axpy operations. For the PPMD implementation it also includes the cost of the short range Lennard-Jones interaction. For each case, the cumulative time spent in the FMM setup phase (Algorithm 2) and the evaluation phase (Algorithm 3) is given, together with the percentage of time spent in the FMM calculation. All times are measured in seconds.

used to simulate a charge-neutral system with periodic boundary conditions and an additional short-range Lennard-Jones interaction), we believe this allows a sensible assessment of the relative performance of the codes. The results in Tab. 7 confirm that in both cases the runtime is dominated by the FMM algorithm. To allow a fair comparison, the parameters were tuned such that both simulations are carried out at comparable accuracy. The ScalFMM code uses the uniform Lagrange interpolation kernel, which depends on the number ℓ_{Lag} of terms. As argued in [55] and confirmed in by the ScalFMM developers [56], setting $\ell_{\text{Lag}} = 5$ is expected to give a relative accuracy of around $\epsilon = 10^{-5}$ in energy and force calculations. As shown in [50], the same relative accuracy is achieved by using $p = 10$ multipole expansion terms in our code. All runs were carried out on a single, fully utilised, Ivy Bridge node. While the ScalFMM code was run in OpenMP mode with 16 threads, results on both a pure MPI run and a mixed mode MPI/OpenMP configuration (2 MPI processes with 8 threads each) are reported for our code. Tab. 7 shows measured runtimes for $N = 10^6$ and $N = 4 \cdot 10^6$ particles. For our implementation the optimal number of levels turned out to be $L = 5$ for $N = 10^6$ and $L = 6$ for $N = 4 \cdot 10^6$, whereas the ScalFMM code gave the best results if the equivalent octree depth was set to 6 in both cases. The results in Tab. 7 confirm that the performance of our code is of the same order of magnitude as the highly optimised ScalFMM library. In fact, the pure MPI implementation is only around 3× slower, which is acceptable given that we have not yet considered further optimisation. Further results on the parallel scalability of our standard FMM implementation can be found in Appendix D.

7. Conclusions

In this paper we presented a new variant of the Fast Multipole Method, which allows the efficient and accurate treatment of electrostatic interactions in kinetic Monte Carlo simulation. Although a recent publication [19] claimed that this would not be possible, we demonstrated that our algorithm scales linearly with the number of charges. This was confirmed numerically by measuring the time t_{step} per KMC step for systems with up to $1.3 \cdot 10^8$ charges. Running in parallel on 8192 cores we find $t_{\text{step}} = 7.09$s. We also presented results for a physically relevant α-NPD test case with 20412 particles which could be simulated at a rate of 0.35s per KMC step on a single 12-core Skylake CPU.

By facilitating the simulation of much larger systems with realistic electrostatics, our new library will allow step-changes in the KMC simulation of energy materials. While the focus of the paper is on the description of the algorithm, its implementation and parallel performance results, a forthcoming publication [54] discusses further results for physically relevant systems.

Our code provides an intuitive user interface and we showed that code generation techniques guarantee excellent scalability and performance on modern HPC installations. In principle the code is performance portable, and has so far been implemented for CPU chips, using hybrid MPI+OpenMP parallelisation. A GPU backend is currently being developed.

An interesting line of future work will be to combine our improved FMM algorithm with novel KMC approaches, such as multilevel KMC techniques. Those methods allow more efficient simulations by skipping physically less interesting transitions, such as "rattling" the frequent repeated hops between pairs of states.

By making suitable adaptations to the FMM algorithm, it will be also possible to reduce the cost of *standard* Monte Carlo (MC) simulations. Similar improvements have already been studied for Ewald summation [13, 27], for which the change in electrostatic energy *per MC move* 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 at each MC step, 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 [57], though the implementation at present is $O(N)$. We believe that a suitably modified FMM algorithm will improve on this and limit the computational complexity per individual MC move to $\mathcal{O}(\log(N))$. The key idea is to store the local expansion $\Psi_{\ell,i}$ on each level of the grid hierarchy, instead of accumulating it on the finest level in the downward sweep. While calculation of the electrostatic field requires evaluation of the $\Psi_{\ell,i}$ in one cell on each of the $L \sim \log(N)$ levels, updating the field only requires changes to a constant number of cells per level. Overall, both operations can be carried out in $\mathcal{O}(L) = \mathcal{O}(\log(N))$ time per MC update.

Acknowledgements

This research made use of the Balena High Performance Computing (HPC) Service at the University of Bath and the Isambard UK National Tier-2 HPC Service (http://gw4.ac.uk/isambard/). Isambard is operated by GW4 and the UK Met Office, and funded by EPSRC (EP/P020224/1). This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreements No 646176 and No 824158. William Saunders was funded by an EPSRC studentship during his PhD. We would like to thank Bérenger Bramas and Olivier Coulaud for help with running ScalFMM on our cluster.

Appendix A. Standard FMM algorithm

For reference the standard FMM algorithm described in Section 2.2 is written down in Algorithms 2 and 3. In addition to the expansions $\Psi_{\ell,\alpha}$ and $\Phi_{\ell,\alpha}$, this requires another p-term local expansion $\bar{\Psi}_{\ell,\alpha}$ about the centre of cell α on level ℓ. $\bar{\Psi}_{\ell,\alpha}$ describes the potential induced by all charges outside the parent of the cell and outside the 26 nearest neighbours of this parent. Algorithm 2 also uses the notion of an *interaction list*

(IL) of a particular cell. This is important to recursively include contributions from finer levels (see line 18 in Algorithm 2). For a cell α on level ℓ the 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 given as

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

On a particular level ℓ the local representation of the operators $\mathcal{T}_{\mathrm{MM}}$ for converting between multipole expansions between cells α and β is given by $\mathcal{T}_{\mathrm{MM}}^{(\ell;\alpha,\beta)}$ with corresponding notation for $\mathcal{T}_{\mathrm{ML}}$ and $\mathcal{T}_{\mathrm{LL}}$.

Algorithm 2 Fast Multipole Algorithm I. Construct local expansion $\Psi_{L,\alpha}$ of long range contribution.

1: **for all** cells $\alpha = 1, \ldots, M = 8^L$ **do**
2: Construct multipole expansion $\Phi_{L,\alpha}$ of all charges contained in cell α
3: **end for**
Upward pass:
4: **for all** levels $\ell = L-1, \ldots, 1$ **do**
5: **for all** cells $\alpha = 1, \ldots, M_\ell = 8^\ell$ **do**
6: Set $\Phi_{\ell,\alpha} = 0$
7: **for all** cells $\beta \in \mathrm{children}(\alpha)$ **do**
8: $\Phi_{\ell,\alpha} \mapsto \Phi_{\ell,\alpha} + \mathcal{T}_{\mathrm{MM}}^{(\ell;\alpha,\beta)} \Phi_{\ell+1,\beta}$
9: **end for**
10: **end for**
11: **end for**
Downward pass:
12: **for all** level $\ell = 2, \ldots, L$ **do**
13: **for all** cells $\alpha = 1, \ldots, M_\ell = 8^\ell$ **do**
14: $\overline{\Psi}_{\ell,\alpha} \mapsto \mathcal{T}_{\mathrm{LL}}^{(\ell;\alpha,\beta)} \Psi_{\ell-1,\beta}$ for $\beta = \mathrm{parent}(\alpha)$
15: **end for**
16: **for all** cells $\alpha = 1, \ldots, M_\ell = 8^\ell$ **do**
17: $\Psi_{\ell,\alpha} \mapsto \overline{\Psi}_{\ell,\alpha}$
18: **for all** cells $\beta \in \mathrm{IL}(\alpha)$ **do**
19: $\Psi_{\ell,\alpha} \mapsto \Psi_{\ell,\alpha} + \mathcal{T}_{\mathrm{ML}}^{(\ell;\alpha,\beta)} \Phi_{\ell,\beta}$
20: **end for**
21: **end for**
22: **end for**

Appendix B. Dipole correction

As discussed in Section 2.2.2, care has to be taken if the dipole moment of the charge distribution is non-zero for periodic- or Dirichlet boundary conditions. Here we derive the correction term which needs to be added to the electric field to enforce the zero surface-charge boundary condition at infinity. First observe that due to the nature of the multipole expansion and the linearity of electrostatics, it is sufficient to derive the term for one particular charge configuration with a given dipole moment. Assume that there is a surface charge density of $+\sigma$ on the top face of the domain, and an opposite density of $-\sigma$ at the opposite face (see Fig. B.12). This induces a dipole moment per unit volume of $p = \sigma$. Let $\phi^{(\square)}$ be the potential which is induced by the primary cell and its 26 neighbours, i.e. the near-field contribution. The corresponding far-field contribution is denoted by $\phi^{(\infty)}$. Calculation of $\phi^{(\square)}$ at the point $\boldsymbol{r} = (x, y, z)$ is straightforward, since we only need to compute the potential generated by two oppositely charged plates of

Algorithm 3 Fast Multipole Algorithm II. Evaluate long- and short- range contributions to potential energy U.

Set $U = 0$
for all cells $\alpha = 1, \ldots, M$ **do**
  for all charges i in cell α **do**
    $U \leftarrow\!\shortmid U + \frac{1}{2}\Psi_{L,\alpha}\left(\boldsymbol{r}^{(i)}\right)$
    for all charges $j \neq i$ in $\alpha \cup \mathcal{N}_b(\alpha)$ **do**
      $U \leftarrow\!\shortmid U + \frac{1}{2}q^{(i)}q^{(j)}/\left|\boldsymbol{r}^{(i)} - \boldsymbol{r}^{(j)}\right|$
    end for
  end for
end for

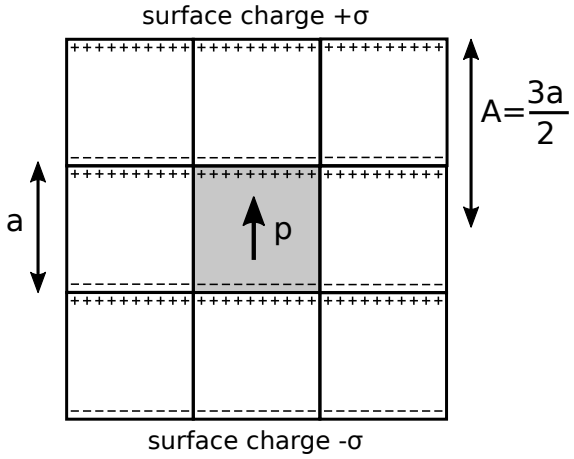

Figure B.12: Charge distribution for working out dipole correction. The primary image is shown in gray. Note that the charges cancel at all interior interfaces, leaving surface charges at the top- and bottom boundaries.

size $(3/2a)^2$ at a distance of $A = 3/2a$ from the origin. The fundamental solution of the Poisson equation and the superposition principle gives

$$\phi^{(\square)} = \int_{-A}^{A}\int_{-A}^{A} \frac{\sigma \, d\eta \, d\xi}{\sqrt{(x-\eta)^2 + (y-\xi)^2 + (A-z)^2}} - \int_{-A}^{A}\int_{-A}^{A} \frac{\sigma \, d\eta \, d\xi}{\sqrt{(x-\eta)^2 + (y-\xi)^2 + (A+z)^2}}$$

We want to calculate the electric field E at the origin, which is given by Taylor-expanding the total potential

$$\phi(\boldsymbol{r}) = \phi^{(\square)}(\boldsymbol{r}) + \phi^{(\infty)}(\boldsymbol{r}) = Ez + \dots .$$

(note that due to symmetry, for this particular setup there are no contributions which are linear in x or y). Since the coefficient $R_2^m = 0$ of the matrix $\mathcal{R}$ introduced in [29] is zero, there is no contribution to E from $\phi^{(\infty)}$. This implies that E can be obtained by taking the derivative of $\phi^{(\square)}$ at the origin. The resulting surface integral is readily evaluated to obtain

$$E = \frac{\partial \phi^{(\square)}}{\partial z}|_{\boldsymbol{r}=0} = 2A\sigma \int_{-A}^{A}\int_{-A}^{A} \left(A^2 + \eta^2 + \xi^2\right)^{-3/2} \, d\eta \, d\xi = \frac{4\pi}{3}\sigma.$$

The same argument can be applied for dipoles pointing in the x- and y- direction; recalling that $\sigma = p$, this implies that the field of a vector-valued dipole density $\boldsymbol{p}$ is given by

$$\boldsymbol{E} = \frac{4\pi}{3}\boldsymbol{p}.$$

Appendix C. Improved interface for proposals

While the `propose()` method described in Section 5.2.1 is intuitive and easy to use, it is not optimal in terms of efficiency. This is because the tuple of proposed moves $\mathcal{P}$ in Eq. (15) has to be converted to an internal data structure before the moves can be passed to our C implementation. To overcome this issue we provide an alternative interface which is more efficient, but requires additional work from the user since the corresponding `propose_with_dats()` method expects the data to be given in a particular, structured format: the proposed moves have to be encoded as a set of particle properties which are stored in `ParticleDat` instances. In contrast to the `propose()` interface, which can operate on a subset of particles, potential new positions have to be specified for all particles in the system. For this, the particles are separated into M different types, such that a particle of type $t \in \{1, \dots, M\}$ can potentially transition to $c_t \in \mathbb{N}_0$ new positions. Note that c_t can be zero, and particles of this type are not able to move at all. The type of a particle could for example depend on the topology of the lattice or the local environment of the lattice site it currently occupies. In this case the type can change during the simulation; an example is described in Appendix C.1. The set $\mathcal{C} = \{c_1, c_2, \dots, c_M\}$ is represented by a M-dimensional `ScalarArray`. The types of all particles are stored in an integer-valued `ParticleDat` $\mathcal{T}$, such that $\mathcal{T}^{(i)} \in \{1, \dots, M\}$ is the type of particle i, which can therefore hop to $c_{\mathcal{T}^{(i)}}$ potential new sites. Let $K = \max(\mathcal{C})$ be the maximum number of moves at any site. An additional real-valued `ParticleDat` $\mathcal{R}$ with $3K$ components is used to store the locations of the potential destinations of all particles. For particle i the entry $\mathcal{R}^{(i)}$ contains the list $\{\boldsymbol{r'}_1^{(i)}, \boldsymbol{r'}_2^{(i)}, \dots, \boldsymbol{r'}_K^{(i)}\}$ of three dimensional vectors where any entries $\boldsymbol{r'}_k^{(i)}$ with $k > c_{\mathcal{T}^{(i)}}$ are irrelevant and may contain arbitrary values. Depending on the local environment of a particle and other particles in its vicinity, particular hops might be blocked for a particular configuration. To avoid changing the type of those particles and shuffling around the entries of $\mathcal{R}^{(i)}$ to account for this, certain transitions can be marked as "forbidden" by setting a flag in a separate `ParticleDat` $\mathcal{M}$ with K components. Each entry $\mathcal{M}^{(i)}$ contains a list $\{m_1^{(i)}, m_2^{(i)}, \dots, m_K^{(i)}\}$ where the flags $m_k^{(i)} = 1$ signals that the transition $\boldsymbol{r'}_k^{(i)} \leftarrow \boldsymbol{r}^{(i)}$ for particle i is allowed. If $m_k^{(i)} < 1$ this transition is forbidden and will not be considered in the calculation of energy changes for the proposed moves.

The `ScalarArray` $\mathcal{C}$ and the `ParticleDats` $\mathcal{T}$, $\mathcal{R}$, $\mathcal{M}$ and $\mathcal{U}$ are passed as inputs to the `propose_with_dats()` method, which populates the `ParticleDat` $\mathcal{U}$ of K components with the energy changes of all proposed

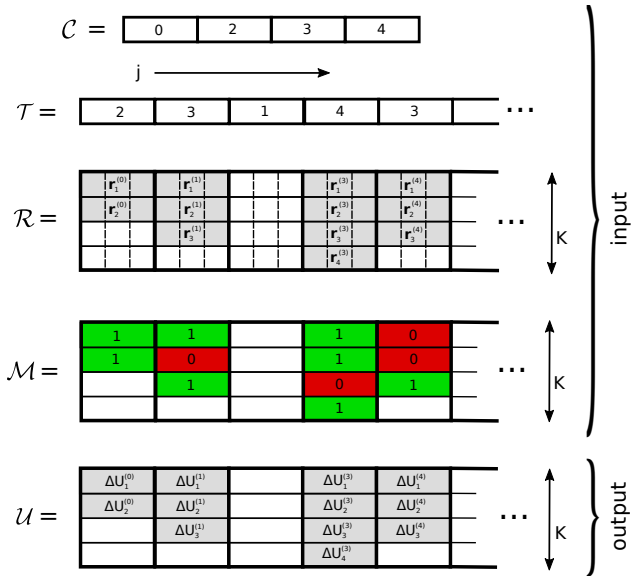

Figure C.13: Data structures (`ScalarArray` $\mathcal{C}$ and `ParticleDats` $\mathcal{T}$, $\mathcal{R}$, $\mathcal{M}$ and $\mathcal{U}$) used by the `propose_with_dats()` interface for proposing potential moves.

moves. More specifically, the entry for particle i is the list $\mathcal{U}^{(i)} = \{\Delta U_1^{(i)}, \Delta U_2^{(i)}, \ldots, \Delta U_K^{(i)}\}$ such that $\Delta U_k^{(i)}$ contains the change in electrostatic energy for the move $\boldsymbol{r}'^{(i)}_k \leftarrow \boldsymbol{r}^{(i)}$. Entries for moves which are marked as forbidden through the mask $\mathcal{M}^{(i)}$ or for which $k > c_{\mathcal{T}^{(i)}}$ are undefined.

An example of the `ParticleDats` $\mathcal{T}$, $\mathcal{R}$, $\mathcal{M}$ and $\mathcal{U}$ is shown in Fig. C.13. In this case each particle can hop to between two and four sites, or not hop at all. The set which described the four different types of particles is therefore given by $\mathcal{C} = \{0, 2, 3, 4\}$ with $K = \max(\mathcal{C}) = 4$.

Using the `propose_with_dats()` and data structures provided by PPMD allows the entire KMC implementation to be written in the looping operations of PPMD. Apart from guaranteeing the efficiency of code, the user never has to explicitly insert parallelisation calls. To illustrate how this can be done, we now show how the inputs to the `propose_with_dats()` method can be set up in PPMD for a particular use case.

Appendix C.1. Selection of allowed moves in PPMD

For this example we assume that the system consists of charges which can hop between the sites of a regular two-dimensional lattice Λ with spacing h embedded in three dimensional space

$$\Lambda = \{\boldsymbol{x} = h\boldsymbol{n} : \boldsymbol{n} \in \mathbb{Z}^2 \times \{0\}, |\boldsymbol{x}| < \frac{1}{2}a\}$$

Recall that the simulation domain is a box of width a, and periodic boundary conditions are assumed for the electrostatic potential. However, we assume that the charges can not hop across the domain boundary. The total number of charges N is assumed to be much smaller than the total number of sites and each site can be occupied by at most one charge. In this example we further assume that charges can only hop to directly neighbouring sites. Note that the number of sites a charge can hop to, i.e. its type, depends on whether it is in the interior of the domain or on the boundary, see Fig. C.14. When setting up the input for `propose_with_dats()`, the following points have to be taken into account:

- All charges need to be be assigned a type by setting the entries in the `ParticleDat` $\mathcal{T}$. This depends on the lattice site the particle currently occupies.
- The potential destinations for all particles have to be worked out in each KMC step by populating the `ParticleDat` $\mathcal{R}$.

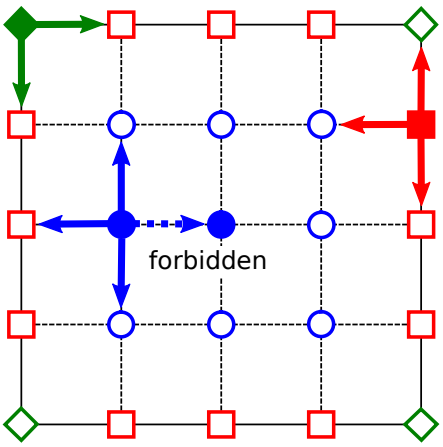

Figure C.14: Two dimensional grid used for the example in Appendix C.1. Sites of different categories (interior, edge, corner) are marked by different symbols and occupied sites are shown as filled.

- Potential hops to already occupied sites need to be masked by setting the entries of the `ParticleDat` $\mathcal{M}$; again this has to be done in each KMC step.

The sites can be arranged into nine different categories (one interior, one for every outer edge/corner of the domain, see Fig. C.14). Since particles sites in the interior of the domain have four direct neighbours, sites on the edges have four and sites in the corners only two, we set

$$\mathcal{C} = \{c_1, c_2, \ldots, c_9\} = \{4, 3, 3, 3, 3, 2, 2, 2, 2\}.$$

Further, for each site category $t \in \{1, \ldots, 9\}$ we define an ordered set of integer offsets

$$\mathcal{D}^{(t)} = \{\boldsymbol{\delta}_1^{(t)}, \boldsymbol{\delta}_2^{(t)}, \ldots, \boldsymbol{\delta}_{c_t}^{(t)}\} \subset \mathbb{Z}^3$$

which describes the potential relative hops of a particle located at a site of this category in units of the lattice spacing,

$$\begin{aligned}
\mathcal{D}^{(1)} &= \{(+1,0,0), (-1,0,0), (0,+1,0), (0,-1,0)\},\\
\mathcal{D}^{(2)} &= \{(+1,0,0), (-1,0,0), (0,+1,0)\},\\
\mathcal{D}^{(3)} &= \{(+1,0,0), (-1,0,0), (0,-1,0)\},\\
&\ldots
\end{aligned}$$

Now consider the charge with index i, which is of type $t = \mathcal{T}^{(i)}$ and currently located at position $\boldsymbol{r}^{(i)} \in \Lambda$. Since the number of potential destinations is c_t, this charge can potentially hop to any point in the set

$$\mathcal{R}^{(i)} = \{\boldsymbol{r}^{(i)} + h\boldsymbol{\delta}_1^{(t)}, \boldsymbol{r}^{(i)} + h\boldsymbol{\delta}_2^{(t)}, \ldots, \boldsymbol{r}^{(i)} + h\boldsymbol{\delta}_{c_t}^{(t)}\} \subset \Lambda.$$

This can be implemented by updating the entries $\mathcal{R}^{(i)}$ of the `ParticleDat` $\mathcal{R}$ with a `ParticleLoop` in the PPMD code.

Finally, forbidden moves to already occupied sites have to be masked by setting appropriate flags in $\mathcal{M}$. This is done by considering all pairs (i, j) of particles and setting the entry $m_k^{(i)}$ of $\mathcal{M}^{(i)}$ to zero if the k-th entry of $\mathcal{R}^{(i)}$ is identical to the current position $\boldsymbol{r}^{(j)}$ of the other particle in the pair, i.e. if $|\boldsymbol{r'}_k^{(i)} - \boldsymbol{r}^{(j)}| = 0$. In PPMD this operation can be realised with a `PairLoop`.

The pseudocode in Algorithm 4 provides an overview of the PPMD implementation of the book-keeping operations discussed in this section. To set the types of the particles it is assumed that there is a function T which returns the category of a lattice site located at $\boldsymbol{r}$.

Algorithm 4 Overview of bookkeeping operations for updating the `ParticleDats` $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{M}$ in each KMC step, as described in Appendix C.1.

Set types & proposed moves (`ParticleLoop`)
for all charges $i = 1, \ldots, N$ **do**
  Set particle types based on position
  $\mathcal{T}^{(i)} \leftarrow\!\shortmid T(\boldsymbol{r}^{(i)})$
  Set proposed positions and initialise masks
  for $k = 1, \ldots, c_{\mathcal{T}^{(i)}}$ **do**
    $\mathcal{R}_k^{(i)} \leftarrow\!\shortmid \boldsymbol{r}^{(i)} + h\boldsymbol{\delta}_k^{\mathcal{T}^{(i)}}$
    $\mathcal{M}_k^{(i)} = 1$
  end for
end for
Detect overlaps (`PairLoop`)
for all pairs (i, j) s.t. $|\boldsymbol{r}^{(i)} - \boldsymbol{r}^{(j)}| == 1$ **do**
  for $k = 1, \ldots, c_{\mathcal{T}^{(i)}}$ **do**
    if $|\boldsymbol{r}'^{(i)}_k - \boldsymbol{r}^{(j)}| == 0$ **then**
      Flag proposed position as conflict
      $\mathcal{M}_k^{(i)} = 0$
    end if
  end for
end for

Appendix D. Parallel Performance of standard FMM

To complement the results in [26] and since our standard FMM implementation in itself might be of interest to others, we compare its performance and parallel scalability with the FFT accelerated Smooth Particle Mesh Ewald (SPME) approach in DL_POLY_4. Here we use a configuration which is based on the two ion NaCl "TEST01" [58] scenario from the DL_POLY test suite. The system is stabilised by adding a repulsive short range Lennard-Jones potential with a cutoff of 4Å. Due to this small cutoff the additional cost of the Lennard-Jones force calculation can be neglected in the reported runtimes. The initial configuration is a cubic lattice of alternating particle species with a lattice spacing of 3.3Å. To allow a fair comparison, for both implementations the parameters were adjusted such that both methods give comparable relative errors of $\sim 10^{-6}$ on the total energy of the system; this required 10 expansion terms in the FMM implementation. For more details on the setup and quantification of errors see [50].

As for the results in Section 6.3, all runs were carried out on the Intel Ivy Bridge E5-2650v2 nodes of the "Balena" HPC facility. In contrast to the setup in Tab. 1, the FMM code was run in two modes:

- A pure MPI implementation, using distributed memory parallelism between the 16 cores of each node.
- A hybrid MPI+OpenMP mode with one MPI rank and 8 OpenMP threads per socket (2×8 OpenMP threads per 16-core node).

DL_POLY was always run in pure MPI mode.

Appendix D.1. Strong scaling

To test the strong scalability we perform 200 Velocity Verlet integration steps of two systems containing $N = 10^6$ and $N = 4.0 \cdot 10^6$ charged particles respectively. The absolute runtimes in Fig. D.15 (left) demonstrate that the performance of our FMM implementation is in the same ballpark as the SPME algorithm in the mature DL_POLY code, which is approximately $3\times$ faster. For larger core counts our FMM implementation does not exhibit unreasonable performance degradation; in fact it scales slightly better that the DL_POLY code. This is further quantified by plotting the strong parallel scalability calculated according

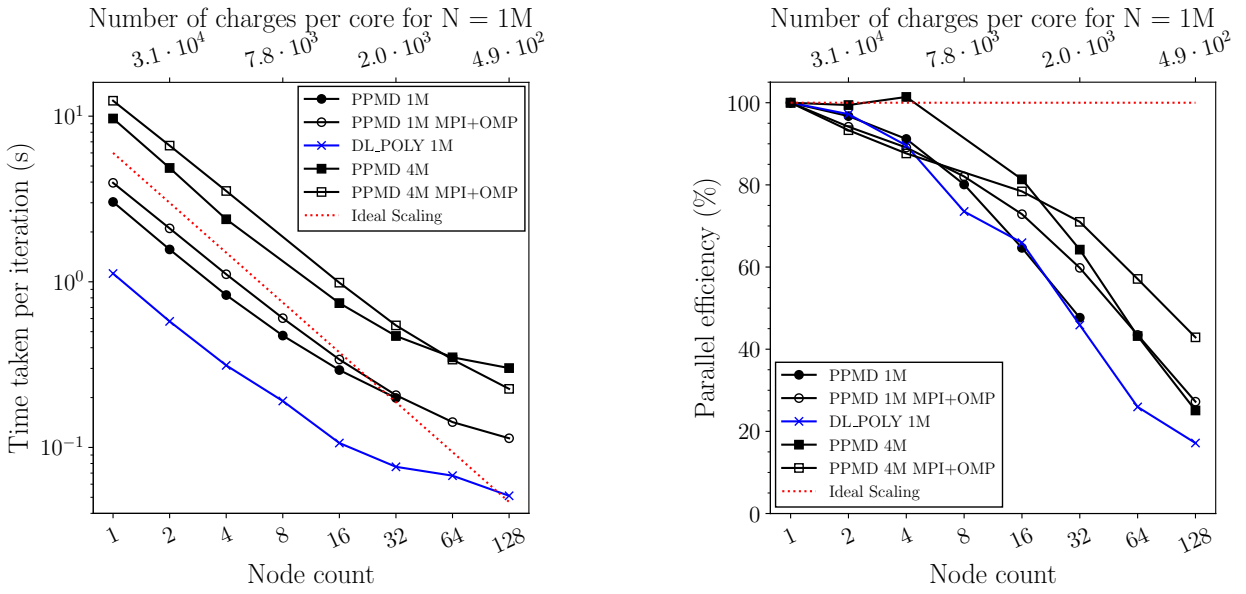

Figure D.15: Strong scaling comparison between our FMM implementation (labelled as "PPMD") and DL_POLY FFT based SPME method. The time per Velocity Verlet step is shown for systems containing $N = 10^6$ (1M) and $N = 4.0 \cdot 10^6$ (4M) charges on the left. The strong parallel efficiency $E_S(P; N)$ as defined in Eq. (20)) is shown on the right.

to Eq. (20) in Fig. D.15 (right). Running in MPI+OpenMP mode further increases parallel efficiency in the strong scaling limit. For smaller node counts, the efficiency of the hybrid approach is poorer than a for pure MPI setup. This is because OpenMP parallelisation introduces atomic operations not found in the distributed memory implementation. Those operations lead to reduced intra-node parallel efficiency, consistent with Amdahl's Law [59].

Appendix D.2. Weak Scaling

In the corresponding weak scaling experiment the number $N_{\text{local}} = 10^6$ of charges per node is kept fixed, while the total number of charges $N = P \cdot N_{\text{local}}$ increases in proportion to the number of nodes P. Since the computational complexity of the FMM algorithm is proportional to N, we expect the time per FMM evaluation to be independent of P. Fig. D.16 (left) shows the time per Velocity Verlet step for total problem sizes between $N = 10^6$ and $N = 1.28 \cdot 10^8$. Due to memory inefficiencies in non-FMM related portions of code it was not possible to run the pure MPI implementation of the code on more than 32 nodes, and we only report results for the hybrid MPI+OpenMPI setup in this case. For each problem size the number of levels L is adjusted to achieve optimal performance. The parallel efficiency $E(P; N_{\text{local}})$ defined in Eq. (21) is plotted in Fig. D.16 (right). As expected, the time per Velocity Verlet step grows only slowly as the number of processors increases.

We refer the interested reader to [50] for a further discussion of those results.

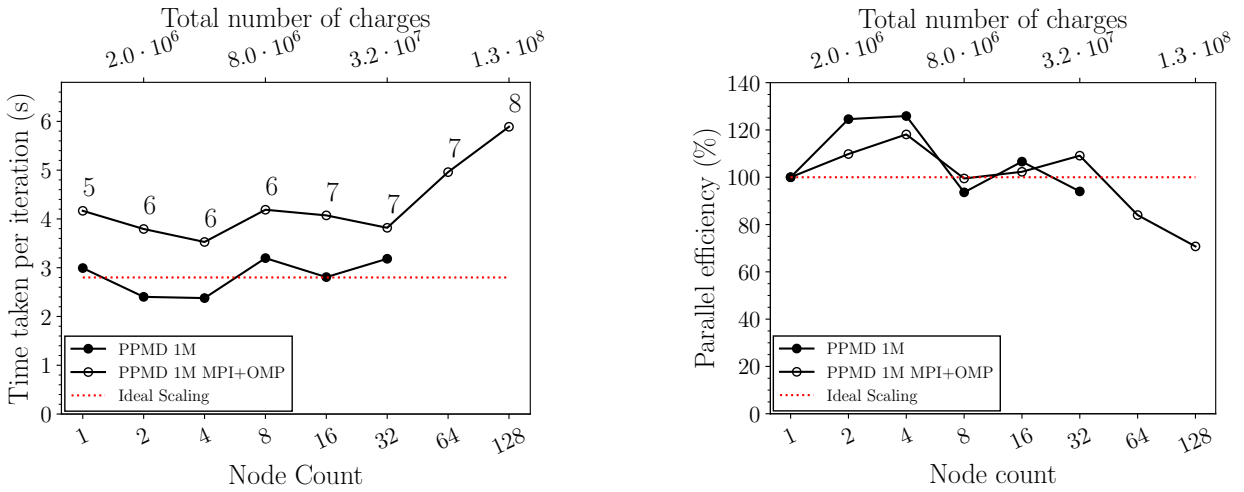

Figure D.16: Weak scaling of the time per Velocity Verlet step for the standard FMM implementation. The absolute time as a function of the number of nodes is shown on the left where floating numbers indicate the number of levels in the octal tree. The parallel efficiency as defined in Equation (21) is shown on the right.

[1] W. Young, E. Elcock, Monte Carlo studies of vacancy migration in binary ordered alloys: I, Proceedings of the Physical Society 89 (3) (1966) 735. doi:10.1088/0370-1328/89/3/329.

[2] A. Bortz, M. Kalos, J. Lebowitz, A new algorithm for Monte Carlo simulation of Ising spin systems, Journal of Computational Physics 17 (1) (1975) 10 – 18. doi:10.1016/0021-9991(75)90060-1.

[3] D. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, Journal of Computational Physics 22 (4) (1976) 403–434. doi:10.1016/0021-9991(76)90041-3.

[4] D. Gillespie, Exact Stochastic Simulation of Coupled Chemical Reactions, Journal of Physical Chemistry 81 (25) (1977) 2340–2361. doi:10.1021/j100540a008.

[5] H. M. Cuppen, L. J. Karssemeijer, T. Lamberts, The Kinetic Monte Carlo Method as a Way To Solve the Master Equation for Interstellar Grain Chemistry, Chemical Reviews 113 (12) (2013) 8840–8871. doi:10.1021/cr400234a.

[6] B. J. Morgan, Lattice-geometry effects in garnet solid electrolytes: A lattice-gas Monte Carlo simulation study, Royal Society Open Science 4 (11) (2017) 170824. doi:10.1098/rsos.170824.

[7] C. Groves, Simulating charge transport in organic semiconductors and devices: A review, Reports on Progress in Physics 80 (2) (2016) 026502. doi:10.1088/1361-6633/80/2/026502.

[8] I. R. Thompson, M. K. Coe, A. B. Walker, M. Ricci, O. M. Roscioni, C. Zannoni, Microscopic origins of charge transport in triphenylene systems, Physical Review Materials 2 (6) (2018) 064601. doi:10.1103/PhysRevMaterials.2.064601.

[9] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, The journal of Chemical Physics 21 (6) (1953) 1087–1092. doi:10.1063/1.1699114.

[10] W. K. Hastings, Monte Carlo sampling methods using Markov chains and their applicationsdoi:10.2307/2334940.

[11] L. Meng, Y. Zhang, X. Wan, C. Li, X. Zhang, Y. Wang, X. Ke, Z. Xiao, L. Ding, R. Xia, H.-L. Yip, Y. Cao, Y. Chen, Organic and solution-processed tandem solar cells with 17.3% efficiency, Sciencedoi:10.1126/science.aat2612.

[12] M. Peplow, Perovskite progress pushes tandem solar cells closer to market, Chemical and Engineering News 96 (24).
URL https://cen.acs.org/energy/solar-power/Perovskite-progress-pushes-tandem-solar/96/i24

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

[14] R. W. Hockney, J. W. Eastwood, Computer simulation using particles, CRC Press, 1988.

[15] M. Deserno, C. Holm, How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines, The Journal of Chemical Physics 109 (18) (1998) 7678–7693. doi:10.1063/1.477414.

[16] 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.

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

[18] 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.

[19] H. Li, J.-L. Brédas, Modeling of Actual-Size Organic Electronic Devices from Efficient Molecular-Scale Simulations, Advanced Functional Materials 28 (29) (2018) 1801460. doi:10.1002/adfm.201801460.

[20] M. Casalegno, G. Raos, R. Po, Methodological assessment of kinetic Monte Carlo simulations of organic photovoltaic devices: The treatment of electrostatic interactions, The Journal of Chemical Physics 132 (9) (2010) 094705. doi:10.1063/1.3337909.

[21] J. Hermet, F. Bottin, G. Dezanneau, G. Geneste, Kinetic Monte Carlo study of protonic diffusion and conduction in Gd-doped BaCeO3, Solid State Ionics 252 (2013) 48–55. doi:10.1016/j.ssi.2013.06.001.

[22] H. Li, J.-L. Brédas, Kinetic Monte Carlo modeling of charge carriers in organic electronic devices: Suppression of the self-interaction error, The Journal of Physical Chemistry Letters 8 (11) (2017) 2507–2512. doi:10.1021/acs.jpclett.7b01161.

[23] J. Van der Holst, F. Van Oost, R. Coehoorn, P. Bobbert, Monte Carlo study of charge transport in organic sandwich-type single-carrier devices: Effects of Coulomb interactions, Physical Review B 83 (8) (2011) 085206. doi:10.1103/PhysRevB.83.085206.

[24] P. Kordt, J. J. van der Holst, M. Al Helwi, W. Kowalsky, F. May, A. Badinski, C. Lennartz, D. Andrienko, Modeling of organic light emitting diodes: From molecular to device properties, Advanced Functional Materials 25 (13) (2015) 1955–1971. doi:10.1002/adfm.201403004.

[25] 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.

[26] 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.

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

[28] M. P. Allen, D. J. Tildesley, Computer simulation of liquids, Oxford University Press, Oxford, 1989.

[29] T. Amisaki, Precise and efficient Ewald summation for periodic fast multipole method, Journal of Computational Chemistry 21 (12) (2000) 1075–1087. doi:10.1002/1096-987X(200009)21:12<1075::AID-JCC4>3.0.CO;2-L.

[30] T. J. Walls, Kinetic Monte Carlo simulations of perovskite crystal growth with long range Coulomb interactions, Ph.D. thesis, College of William and Mary. (1999).

[31] U. Trottenberg, C. W. Oosterlee, A. Schuller, Multigrid, Elsevier, 2000.

[32] N. van der Kaap, L. J. A. Koster, Massively parallel kinetic Monte Carlo simulations of charge carrier transport in organic semiconductors, Journal of Computational Physics 307 (2016) 321–332. doi:10.1016/j.jcp.2015.12.001.

[33] P. Blanchard, B. Bramas, O. Coulaud, E. Darve, L. Dupuy, A. Etcheverry, G. Sylvand, ScalFMM: A generic parallel fast multipole library, in: SIAM Conference on Computational Science and Engineering (SIAM CSE 2015), 2015.
URL https://hal.inria.fr/hal-01135253

[34] R. Yokota, L. A. Barba, A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems, The International Journal of High Performance Computing Applications 26 (4) (2012) 337–346. doi:10.1177/1094342011429952.

[35] R. Yokota, L. A. Barba, T. Narumi, K. Yasuoka, Petascale turbulence simulation using a highly parallel fast multipole method on GPUs, Computer Physics Communications 184 (3) (2013) 445–455. doi:10.1016/j.cpc.2012.09.011.

[36] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, G. Biros, A massively parallel adaptive fast-multipole method on heterogeneous architectures, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, 2009, p. 58. doi:10.1145/1654059.1654118.

[37] Z. Gimbutas, L. Greengard, Computational software: Simple FMM libraries for electrostatics, slow viscous flow, and frequency-domain wave propagation, Communications in Computational Physics 18 (2) (2015) 516–528. doi:10.4208/cicp.150215.260615sw.

[38] B. Zhang, J. Huang, N. P. Pitsianis, X. Sun, RECFMM: Recursive parallelization of the adaptive fast multipole method for coulomb and screened coulomb interactions, Communications in Computational Physics 20 (2) (2016) 534–550. doi:10.4208/cicp.230216.140416sw.

[39] J. DeBuhr, B. Zhang, A. Tsueda, V. Tilstra-Smith, T. Sterling, Dashmm: Dynamic adaptive system for hierarchical multipole methods, Communications in Computational Physics 20 (4) (2016) 1106–1126. doi:10.4208/cicp.030316.310716sw.

[40] R. Yokota, Fast Multipole Methods (webpage), https://sites.google.com/site/rioyokota/research/fmm (2015).

[41] D. S. Gunn, N. L. Allan, J. A. Purton, Adaptive kinetic Monte Carlo simulation of solid oxide fuel cell components, Journal of Materials Chemistry A 2 (33) (2014) 13407–13414. doi:10.1039/C4TA01504E.

[42] S. Plimpton, C. Battaile, M. Chandross, L. Holm, A. Thompson, V. Tikare, G. Wagner, E. Webb, X. Zhou, C. G. Cardona, et al., Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo, Sandia Report SAND2009-6226doi:10.2172/966942.

[43] M. Leetmaa, N. V. Skorodumova, KMCLib: A general framework for lattice kinetic Monte Carlo (KMC) simulations, Computer Physics Communications 185 (9) (2014) 2340–2349. doi:10.1016/j.cpc.2014.04.017.

[44] F. Liu, H. van Eersel, B. Xu, J. G. Wilbers, M. P. de Jong, W. G. van der Wiel, P. A. Bobbert, R. Coehoorn, Effect of Coulomb correlation on charge transport in disordered organic semiconductors, Physical Review B 96 (20) (2017) 205203. doi:10.1103/PhysRevB.96.205203.

[45] Simbeyond B.V., Bumblebee code, https://simbeyond.com/bumblebee/.

[46] W. R. Saunders, J. Grant, E. H. Mueller, I. Thompson, Code and Data Release: Fast electrostatic solvers for kinetic Monte Carlo simulations (May 2019). doi:10.5281/zenodo.2677705.
URL https://doi.org/10.5281/zenodo.2677705

[47] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, Journal of Computational Physics 117 (1) (1995) 1 – 19. doi:10.1006/jcph.1995.1039.

[48] W. Smith, T. Forester, DL_POLY_2.0: A general-purpose parallel molecular dynamics simulation package, Journal of Molecular Graphics 14 (3) (1996) 136 – 141. doi:10.1016/S0263-7855(96)00043-4.

[49] I. T. Todorov, W. Smith, K. Trachenko, M. T. Dove, DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism, Journal of Materials Chemistry 16 (2006) 1911–1918. doi:10.1039/B517931A.

[50] W. R. Saunders, Development of A Performance-Portable Framework For Atomistic Simulations, Ph.D. thesis, University of Bath (2018).

[51] M. Abramowitz, I. A. Stegun, Handbook of mathematical functions: with formulas, graphs, and mathematical tables, Vol. 55, Courier Corporation, 1965.

[52] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical recipes 3rd edition: The art of scientific computing, Cambridge university press, 2007.

[53] A. Massé, P. Friederich, F. Symalla, F. Liu, R. Nitsche, R. Coehoorn, W. Wenzel, P. A. Bobbert, Ab initio charge-carrier mobility model for amorphous molecular semiconductors, Physical Review B 93 (2016) 195209. doi:10.1103/PhysRevB.93.195209.

[54] I. R. Thompson, A. B. Walker, W. R. Saunders, O. M. Roscioni, G. DAvino, Effect of long-range interactions on mesoscopic carrier dynamics and organic semiconductor doping efficiency, in preparation.

[55] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, T. Takahashi, Task-based FMM for multicore architectures, SIAM Journal on Scientific Computing 36 (1) (2014) C66–C93. doi:10.1137/130915662.

[56] B. Bramas, O. Coulaud, private communications (2019).

[57] 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.

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

[59] G. M. Amdahl, Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, in: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), ACM, New York, NY, USA, 1967, pp. 483–485. doi:10.1145/1465482.1465560.