GPU fast multipole method with lambda-dynamics features

Dissertation

for the award of the degree

"Doctor rerum naturalium" (Dr.rer.nat)

of the Georg-August-Universität Göttingen

within the doctoral program PCS

of the Georg August University School of Science (GAUSS)

submitted by

Bartosz Kohnke

from Kołobrzeg

Göttingen, 2020

Thesis Committee
Prof. Dr. Gert Lube
Prof. Dr. Helmut Grubmüller

Members of the Examination Board
Reviewer: Prof. Dr. Gert Lube
Second Reviewer: Prof. Dr. Helmut Grubmüller
Additional Reviewer: Prof. Dr. Gundolf Haase

Further members of the Examination Board
Prof. Dr. Ramin Yahyapour
Prof. Dr. Christoph Lehrenfeld
Dr. Johannes Söding
Prof. Dr. Marcus Baum

Date of the oral examination: 24.11.2020

Contents

I INTRODUCTION **1**

I.1 Approximation techniques 2

I.2 FMM complexity 4

I.3 FMM development and parallelization approaches 5

I.4 Parallelization challenges in FMM 5

I.5 FMM as a PME alternative for electrostatic calculations 7

I.6 Project goals 7

II PARALLELIZATION AND ACCURACY / PERFORMANCE EVALUATION OF THE FMM FOR GROMACS **9**

II.1 GROMEX: A scalable and versatile Fast Multipole Method for biomolecular simulation 10

II.2 A CUDA Fast Multipole Method with highly efficient M2L far field evaluation 41

II.3 A GPU-accelerated Fast Multipole Method for GROMACS: performance and accuracy 87

III CONCLUSIONS AND OUTLOOK **121**

III.1 Accuracy of the CUDA FMM 121

III.2 Performance of the CUDA FMM 121

III.3 Parallelization and performance of the λ-FMM 123

III.4 Outlook 124

I Introduction

The n-body problem has been challenging scientists already over three hundred years. Its special case, the gravitational two-body problem, was solved already in 1687 by Sir Isaac Newton and published in his *Principia*. However, it took another two hundred years to find a solution for systems with more than two bodies. In 1885 Gösta Mittag-Leffler arranged an international competition in mathematics in which the most prominent question read as follows:

"*For a system of arbitrarily many mass points that attract each other according to Newtons law, assuming that no two points ever collide, find a series expansion of the coordinates of each point in known functions of time converging uniformly for any period of time.*"

After nearly three years of tackling the three-body problem [26], Henri Poincaré found out that such a system can behave chaotically. It took another 20 years until Karl Sundman found a definitive solution for $n = 3$. Finally, in 1991 Wang Qiu-Dong generalized Sundman's concept to n bodies [42]. However, this solution has no practical meaning because of the "terribly slow" convergence of the resulting series expansion. For that reason, the n-body problem still has to be solved numerically by directly calculating forces acting on all particles in discrete time intervals.

The solution of an n-body problem is an essential task in a variety of scientific fields like astrophysics [41, 2], plasma physics [12], or biomolecular physics [8, 25]. Here, we consider molecular dynamics (MD) simulations, which meanwhile have become a routine to study the dynamics of biomolecules, such as proteins, on timescales inaccessible to experiments. To this aim, MD describes an atomic structure of given molecules by a system of point charges. The dynamics of the complete system is then approximated in terms of Newtonian mechanics. To calculate forces acting on all atoms in the system, the n-body problem is solved. Its result allows to update the atomic positions by integrating the resulting equations of motion. The integration is performed numerically with an explicit method, e.g. the Leapfrog algorithm [18]. Repeating this process in discrete time intervals yields atomic trajectories, which are used to study the molecular behavior. However, to resolve the fastest atomic vibrations the integration time step has to be in a few femtoseconds range. Because many biomolecules operate on a millisecond timescale, this implies a very large number of MD steps to obtain viable atomic trajectories [33, 7, 40].

When considering a single MD step, the overall potential of a given system is described by the force field [36, 21, 35] that incorporates two parts: a bonded and a non-bonded part. The bonded part consists of terms characterizing the forces between covalently bonded atoms. The evaluation of this part requires $\mathcal{O}(n)$ steps. The non-bonded part incorporates Pauli repulsion and van der Waals forces. These are approximated by the Lennard-Jones potential, which is also calculated within $\mathcal{O}(n)$ steps since it decays rapidly with a growing distance. In addition, the non-bonded part comprises the Coulomb potential, which describes forces $\mathbf{f}_i = (f_x, f_y, f_z)$

acting on the i-th particle residing at the position coordinate $\mathbf{x}_i = (x_x, x_y, x_z)$ with

$$\mathbf{f}_i = q_i \sum_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{q_j}{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2} \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|_2}, \quad i = 0, ..., n-1, \tag{1}$$

where $\|\cdot\|_2$ denotes the Euclidean distance and q_i is the charge of the i-th particle. Integrating Eq. 1 yields the electrostatic potential

$$\Phi_i = \sum_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{q_j}{\|\mathbf{x}_i - \mathbf{x}_j\|_2}, \quad i = 0, ..., n-1, \tag{2}$$

evaluated at the target position $\mathbf{x}_i$. The electrostatic energy of the entire system can then be obtained with

$$E = \frac{1}{2} \sum_{i=0}^{n-1} \Phi_i q_i. \tag{3}$$

A direct calculation of Eq. 1 or Eq. 2 requires evaluation of all pairs $(\mathbf{x}_i, \mathbf{x}_j)$ leading to $\mathcal{O}(n^2)$ scaling. This calculation, even on a modern hardware, is only applicable for a moderate number of particles, thus the solution for larger n requires an approximation technique with a better scaling.

A further challenge arises when the determination of titratable states in particular atomic groups is required. Such calculations are described by constant-pH [13] or, more generally, by λ-dynamics algorithms [24, 28, 30]. λ-dynamics introduces a λ variable, which is treated as an additional degree of freedom in the simulations. The algorithm also requires the determination of additional energies, which describe alternative, protonated (A) or deprotonated (B), states of a group. For one group with two states the Eq. 3 is extended by

$$E = (1-\lambda)\frac{1}{2} \sum_{i=0}^{n-1} \Phi_i^A q_i^A + \lambda \frac{1}{2} \sum_{i=0}^{n-1} \Phi_i^B q_i^B, \tag{4}$$

where $0 < \lambda < 1$. The λ variable couples the alternative energies. Its value is obtained by evaluating the force acting on a fictive λ particle, which moves along its own coordinate. The force calculation requires a separate evaluation of each of the right hand side terms of Eq. 2. When using an approximation technique, its ability to perform also this calculation efficiently would be beneficial to speed up λ-dynamics simulations.

I.1 Approximation techniques

One of the simplest schemes to speedup the calculation of the n-body problem is the cut-off technique [37, 10]. It evaluates pairwise interactions within a limited distance $r < R$, where R is a preset cut-off range and $r = \|\mathbf{x}_i - \mathbf{x}_j\|_2$. This improves

the scaling from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. However, the long range nature of the $1/r$ potential leads to insufficient accuracy of this approach [39].

To maintain a desired accuracy and to improve the scaling, we need a better approximation technique. One of the first approaches to approximate Eq. 1 is the Ewald summation [17], proposed in 1921. It is applied to calculate energies in systems with periodic boundary conditions (PBC). The method splits the calculation into two rapidly converging terms: the direct and the reciprocal term. The direct short-range term is evaluated with the cut-off scheme, whereas the reciprocal long-range term is modified algebraically to reduce the overall complexity to $\mathcal{O}(n^{3/2})$. Further enhancements of the Ewald sums led to the particle mesh Ewald (PME) algorithm [16], which is currently the most prominent method in the MD field. It enhances the calculation of the reciprocal space Ewald sums by interpolating the charges on an equally spaced mesh and by evaluating the resulting convolutions using the fast Fourier transform (FFT), which scales with $\mathcal{O}(n \log n)$. As the cut-off part is of order $\mathcal{O}(n)$, the PME scaling is $\mathcal{O}(n \log n)$. The approximation error and performance of the method is controlled by the cut-off radius size, interpolation order and the Fourier mesh spacing. The algorithm, by construction, provides a PBC solution, which is of particular interests for many biomolecular simulations; however, open boundary simulations are not accessible. The method has been optimized over the last few decades and it achieves remarkable performance allowing to evaluate forces and energies within milliseconds for system sizes of $\approx 10^5$ on a single workstation with one graphics processing unit (GPU).

A different approach to approximate Eq. 1 is a multigrid algorithm [9], which was introduced by Brandt as a solver for elliptical partial differential equations. It utilizes a hierarchical grid, on which the equation residual is smoothed to remove high frequency errors. The smoothing employs relaxation methods, which require only few iterations to sufficiently smooth the residual so that it can be recursively transferred to more coarser grids to eliminate all error frequencies. The method scales with $\mathcal{O}(n)$ and it is applicable for both periodic and non-periodic systems. Multigrid can be parallelized efficiently [43], however its use for MD is very limited. The method shows a very good parallel scaling but, in comparison to PME, its performance is slightly worse at required accuracy. Furthermore, it exhibits a large energy drift due to accumulation of different errors [2].

A third group of algorithms to solve the n-body problem are the tree-based methods. The treecode algorithm [1, 3], proposed by Appel and Barnes and Hut in 1986, achieves the $\mathcal{O}(n \log n)$ scaling by approximating the n-body problem by hierarchical grouping of the distant interactions into single interactions. The method partitions the 3D space of a simulation cell by recursively subdividing it into eight subboxes. The subdivision yields an octree, on which the interaction hierarchy is described. Further improvement of the treecode led to a Fast Multipole Method (FMM) [22], which scales linearly with respect to the number of particles n. It was proposed by Greengard and Rokhlin in 1987 and it utilizes spherical harmonics to approximate the r^{-1} potential in terms of local moments, which are constructed by applying linear operators to multipole moments. This transformation has $\mathcal{O}(p^4)$ complexity, where

p is the expansion order. A precise description of the FMM follows in subsequent sections.

I.2 FMM complexity

Execution of the FMM algorithm consists of six different stages. The operations in each stage are performed on data structures and particles contained in octree boxes. The octree subdivision of the 3D computational domain yields $8^{\mathcal{D}}$ boxes on the deepest octree level $\mathcal{D}$. The first stage of the algorithm is Particle-to-Multipole (P2M), which has $\mathcal{O}(p^2)$ complexity. In P2M, the multipole expansions in each box on the level $\mathcal{D}$ are formed. Hence, since all particles contribute to multipoles on the lowest level, the overall complexity of this stage is $\mathcal{O}(np^2)$. Afterwards, the multipole expansions are formed in all coarser level boxes of the octree. This is performed by level-wise translating the expansion centers of the multipoles from finer to coarser tree levels. The operation is called Multipole-to-Multipole (M2M). It has $\mathcal{O}(p^4)$ complexity and it does not directly depend on n. However, the proper choice of $\mathcal{D}$ depends on n. There are

$$\sum_{d=0}^{\mathcal{D}} 8^d = \lfloor \frac{8^{\mathcal{D}+1}}{7} \rfloor = \lfloor \frac{8}{7} 8^{\mathcal{D}} \rfloor \tag{5}$$

boxes in the octree. It can be assumed that $8^{\mathcal{D}} \leq n$, thus $\mathcal{O}(8^{\mathcal{D}}) \in \mathcal{O}(n)$. Hence, the overall complexity of the M2M stage is $8/7 \cdot \mathcal{O}(np^4)$. Subsequently, the multipole expansions are transformed into local expansions within a stage called Multipole-to-Local (M2L). A single M2L transformation has also $\mathcal{O}(p^4)$ complexity and it is performed 189 times for each box in the octree. Therefore, the overall complexity of the M2L stage is $8/7 \cdot 189 \cdot \mathcal{O}(np^4)$. In the last $\mathcal{O}(p^4)$ operation (Local-to-Local), the local expansions are translated from the root to the leaves of the octree. This translation is performed once for each box in the octree, hence, it has $8/7 \cdot \mathcal{O}(np^4)$ overall complexity. The evaluation of the potential is performed for each particle and requires $\mathcal{O}(p^2)$ operations on local moments. This yields $\mathcal{O}(np^2)$ overall complexity of this stage. Additionally, the direct particle interaction stage (P2P) has $27 \cdot \mathcal{O}(n)$ complexity as the calculations are performed only between adjacent boxes on the lowest level $\mathcal{D}$.

Summing the scaling of all FMM stages yields

$$\frac{8}{7} 191 \mathcal{O}(np^4) + 2\mathcal{O}(np^2) + 27\mathcal{O}(n) \tag{6}$$

overall scaling of the method. As p is treated as a constant parameter, which is set to achieve a desired accuracy, Eq. 6 simplifies to

$$\approx 247\, \mathcal{O}(n). \tag{7}$$

I.3 FMM development and parallelization approaches

The originally proposed FMM [22] utilizes $\mathcal{O}(p^4)$ far-field operators. As these operators are the main bottleneck of the algorithm, further approaches to reduce their complexity were developed. The rotational operators [45], which align the z-axis of the transformation were proposed. They reduce the operator complexity to $\mathcal{O}(p^3)$; however, they introduce an additional $\mathcal{O}(p^3)$ operation to rotate the moments. After rotating, the original $\mathcal{O}(p^4)$ transformation reduces to a $\mathcal{O}(p^3)$ transformation as the rotation preserves the azimuthal symmetry of the multipoles along the z-axis. However, the tetrahedron-like shape of the rotational data structure hinders its efficient parallel implementation. The grid-based approach [4] improves the scaling of the operator even further to $\mathcal{O}(p^2)$ but it has a large memory requirement and provides no analytic error bounds. A $\mathcal{O}(p^2 \log p)$ operator, which utilizes FFTs has been proposed but it also has a limiting memory requirement [15]. Further operator enhancements [11] utilize plain-wave expansions to reduce the complexity of a part of the operators to $\mathcal{O}(p^2)$. This approach additionally requires six different $\mathcal{O}(p^3)$ operators to transform multipoles to their plane-wave representation.

A different approach to enhance the performance of the FMM is to employ structure adaptive methods [38, 14]. In contrast to enhancing the operator performance, they require much lower p to achieve a desired accuracy. This improvement is possible by exploiting space decompositions that take into account a structural information of the underlying system of particles.

Furthermore, different parameterizations of the r^{-1} function have been considered. The "black-box FMM" [19] utilizes Chebyshev polynomials, which use a minimal number of coefficients that describe the system potential. Further method developments led to a more general formulation [46], which also supports oscillatory kernels e^{ikr}/r that can not be described by spherical harmonics.

Early GPU FMM implementation [23] utilizing $\mathcal{O}(p^3)$ operators achieved speedups of up to 70 when compared to a serial implementation. One of the latest GPU implementation [20] utilizes GPU concurrent streams to optimize the executions of the $\mathcal{O}(p^3)$ M2L operator on a single GPU. The black-box FMM was also parallelized efficiently on GPUs [44]. MPI based implementations [48, 47] on clusters with on-node GPUs achieved efficiencies of up to 66% for systems with 10^7 particles. Very large multi-node, multi-GPU parallelization with more then 256 GPUs followed [34]. Also task based parallelization approaches of the FMM were undertaken [5].

I.4 Parallelization challenges in FMM

With the rise of modern multi-core, multi-node architectures, where each node can additionally contain several GPUs as computational accelerators, an optimal parallelization has become a tedious task as it requires to study a variety of different hardware bottlenecks. Additionally, tasks performed by various algorithmic parts

typically exhibit a different amount of fine-grained parallelism. This also complicates an efficient single GPU parallelization, because GPUs provide only limited task synchronization tools. When targeting an order of millisecond wall time for a one time step, as required in MD simulations, latency hiding and synchronization become the main bottlenecks in the implementation.

When considering an efficient MD parallelization, an additional challenge is posed by the scaling property of MD simulations. In contrast to many other fields, e.g. weather forecasting or fluid dynamics in which weak scaling is sufficient to achieve more accurate results for a given system, MD simulations require strong scaling as the atomic description and the operational timescales of biomolecules are fixed. Including an additional quantum information or using more sophisticated sampling techniques still requires long trajectories. Hence, minimizing the execution time needed for one MD step, in which the n-body calculation is by far the most limiting factor, is crucial to enhance the overall performance.

The FMM applies a sequence of operators to transform an input of the particle positions into an output of the forces acting on them. The direct calculation part of the FMM, which performs Eq. 1 on a subset of all particles, is a compute-bound, single instruction multiple data (SIMD) task. This task is especially suited for GPUs, which excellently combine the SIMD execution concept with multithreading, referring to it as a single instruction, multiple threads (SIMT). Hence, a GPU implemented direct calculation part of the FMM clearly outperforms even highly optimized SIMD parallelized CPU versions.

The execution order of the far-field operators is inherently sequential but most of the operators provide a large amount of parallelism, which can be divided into two groups: tree and operator parallelism. The tree parallelism exposed by the method enables very efficient large-scale parallelization as it contains a great number of independent operations with limited communication. Considering operator parallelism, the level of parallelism varies strongly depending on a particular operator. A few of the operators are recursively formulated what limits their amenity for efficient GPU parallelization. The most performance limiting far-field operation, multipole-to-local (M2L), provides a lot of tree parallelism with negligible communication requirement. This property leads to a large number of independent operations that can be distributed efficiently on heterogeneous hardware. It can also be exploited for a very good single GPU performance. However, harnessing the operator parallelism requires involved parallelization to achieve an efficient work-load distribution on a single GPU. This is caused by the highly non-uniform data access required for the M2L calculation. Additionally, the operations preformed on these data structures are not part of any highly optimized existing library.

I.5 FMM as a PME alternative for electrostatic calculations

The main drawback of the prevailing PME method is its limitation to massive parallelization because the underlying FFTs require all-to-all communication. For p processes, sending of $\mathcal{O}(p^2)$ messages is required. Such communication overhead is only negligible for a moderate parallelization [6, 32, 31]. Additionally, the memory demand for storing the Fourier mesh increases with growing simulation box size. This property prevents PME to treat very large number of particles or smaller systems with sparsely or not uniformly distributed particles especially on a single computational node. Moreover, PME can not provide an open boundary solution, as the underlying formalism allows only periodic calculations.

The FMM does not suffer from these drawbacks of the PME. The octree structure of the FMM is expected to allow for a very efficient exascale parallelization since the communication required by the method decreases with a distance of interacting particles. Clearly, as the problem sizes are mostly fixed, strong scaling is required to provide long trajectories. Additionally, a single-node FMM implementation requires less memory than PME. The FMM also provides the possibility to compute a open boundary solution or partly periodic solution where the periodicity is taken into account only along a chosen coordinate axis.

Another advantage of the FMM originates from the flexibility of the octree. FMM is able to efficiently calculate alternative energies emerging from different states in λ-dynamics formulation. Such calculation is not trivial with PME, as, to get the required energy for each state, one separate mesh evaluation is needed. Such re-evaluation introduces a substantial runtime overhead, because it requires a separate run of the globally defined FFT. A slightly different approach to utilize PME for the calculation of alternative energies was proposed by Berk Hess with a charge scaling method (internal communication). In contrast to λ-dynamics, which uses λ values to scale hamiltonians of the system, the method directly scales charges of the titratable groups. This leads to a different dynamics of the λ particle but it can be easily implemented with PME. However, for a strict λ-dynamics implementation FMM is advantageous as it is able to compute additional alternative energies with negligible costs by introducing multiple sparse octrees interacting with each other. As the energies emerge mostly due to local changes in particle distribution, most re-calculations are performed only locally on the deepest level of the octrees. The FMM is expected to perform very fast, since the alternative states are typically described by a few particles only.

I.6 Project goals

The main goal of the project is to efficiently implement the spherical harmonics based FMM algorithm for a GPU utilization, to determine the FMM parameters yielding sufficient accuracy for MD simulations, and to evaluate accuracy and performance of the method. Further, bottlenecks of different FMM stages will be

investigated to assess their amenity for a future heterogeneous parallelization so that the method can be then used as an alternative electrostatic solver in the MD software package GROMACS [27] on exascale systems. Additionally, the FMM will be enhanced to allow for an efficient execution of the λ-dynamics algorithm.

II Parallelization and accuracy / performance evaluation of the FMM for GROMACS

Here, we describe our MD optimized CUDA implementation of the FMM. The first section will specify further challenges of heterogeneous parallelization of the FMM, will give an overview of the entire FMM CUDA parallelization process and will show the first scalings and timings. Additionally, the first results of an efficient CUDA-FMM usage for λ-dynamics framework will be reported.

The second subsection will describe the CUDA parallelization of the M2L operator, which is a most performance limiting far-field operator in the FMM. It will illustrate three different parallelization strategies; from a very simple rapid CUDA parallelization approach to a most advanced, symmetry exploiting approach.

The third section will evaluate the performance of the MD software package GROMACS executed with CUDA-FMM electrostatics. This section will also establish adequate FMM parameter to achieve the desired accuracy by comparing FMM and PME errors for various computational scenarios.

II.1 GROMEX: A scalable and versatile Fast Multipole Method for biomolecular simulation

In this work, I provided an overview of the CUDA GPU parallelization process and I showed first preliminary results of the GPU FMM performance. To this aim, I wrote the Section 3.4, which briefly describes the FMM operators and their GPU parallelization, and Additionally, in Section 3.5 I presented first results of the λ-dynamics FMM implementation. Finally, I wrote the corresponding part of the Conclusions and Outlook. The entire content of the paper was subsequently edited by Prof. Dr. Gert Lube and by Prof. Dr. Helmut Grubmüller. The paper has been published in Software for Exascale Computing - SPPEXA 2016–2019 [29].

GROMEX: A scalable and versatile Fast Multipole Method for biomolecular simulation

Bartosz Kohnke[1], Thomas R. Ullmann[1], Andreas Beckmann[2], Ivo Kabadshow[2], David Haensel[2], Laura Morgenstern[2], Plamen Dobrev[1], Gerrit Groenhof[3], Carsten Kutzner[1], Berk Hess[4], Holger Dachsel[2], and Helmut Grubmüller[1]

[1]*Max Planck Institute for Biophysical Chemistry, Theoretical and Computational Biophysics, Am Faßberg 11, 37077 Göttingen*
[2]*Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm Johnen Straße 1, 52428 Jülich*
[3]*Nanoscience Center, Department of Chemistry, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland*
[4]*Department of Physics, Swedish e-Science Research Center, KTH Royal Institute of Technology, Stockholm, Sweden*

Atomistic simulations of large biomolecular systems with chemical variability such as constant pH dynamic protonation offer multiple challenges in high performance computing. One of them is the correct treatment of the involved electrostatics in an efficient and highly scalable way. Here we review and assess two of the main building blocks that will permit such simulations: (i) An electrostatics library based on the Fast Multipole Method (FMM) that treats local alternative charge distributions with minimal overhead, and (ii) A λ-dynamics module working in tandem with the FMM that enables various types of chemical transitions during the simulation. Our λ-dynamics and FMM implementations do not rely on third-party libraries but are exclusively using C++ language features and they are tailored to the specific requirements of molecular dynamics simulation suites such as GROMACS. The FMM library supports fractional tree depths and allows for rigorous error control and automatic performance optimization at runtime. Near-optimal performance is achieved on various SIMD architectures and on GPUs using CUDA. For exascale systems, we expect our approach to outperform current implementations based on Particle Mesh Ewald (PME) electrostatics, because FMM avoids the communication bottlenecks caused by the parallel fast Fourier transformations needed for PME.

1 Introduction

The majority of cellular function is carried out by biological nanomachines made of proteins. Ranging from transporters to enzymes, from motor to signalling proteins, conformational transitions are frequently at the core of protein function, which renders the detailed understanding of the involved dynamics indispensable. Experimentally, atomistic dynamics on submillisecond timescales are notoriously difficult to access, making computer simulations the method of choice. Molecular dynamics (MD) simulations of biomolecular systems are nowadays routinely used to study the mechanisms underlying biological function in atomic detail. Examples reach from membrane channels [28], microtubules [20], and whole ribosomes [4] to subcellular organelles [43]. Recently, the first MD simulation of an entire gene was reported, comprising about a billion of atoms [21].

Apart from system size, the scope of such simulations is limited by model accuracy and simulation length. Particularly the accurate treatment of electrostatic interactions is essential to properly describe a biomolecule's functional motions. However, these interactions are numerically challenging for two reasons.

First, their long-range character (the potential drops off slowly with r^{-1} with distance r) renders traditional cut-off schemes prone to artifacts, such that grid-based Ewald summation methods were introduced to provide an accurate solution in 3D periodic boundaries. The current standard is the Particle Mesh Ewald (PME) method that makes use of fast Fourier transforms (FFTs) and scales as $N \cdot \log N$ with the number of charges N [11]. However, when parallelizing PME over many compute nodes, the algorithm's communication requirements become more limiting than the scaling with respect to N. Because of the involved FFTs, parallel PME requires multiple all-to-all communication steps per time step, in which the number of messages sent between p processes scales with p^2 [29]. For the PME algorithm included in the highly efficient, open source MD package GROMACS [42], much effort has been made to reduce as much as possible the all-to-all bottleneck, e.g. by partitioning the parallel computer in long-range and short-range processors, which reduces the number of messages involved in all-to-all communication [17]. Despite these efforts, however, even for multimillion atom MD systems on modern hardware, performance levels off beyond several thousand cores due to the inherent parallelization limitations of PME [45, 42, 30].

The second challenge is the tight and non-local coupling between the electrostatic potential and the location of charges on the protein, in particular titratable/protonatable groups that adapt their total charge and potentially also their charge distribution to their current electrostatic environment. Hence, all protonation states are closely coupled, depend on pH, and therefore the protonation / deprotonation dynamics needs to be taken into account during the simulation. Whereas most MD simulations employ fixed protonation states for each titratable group, several dynamical schemes have been introduced [37, 33, 23, 8, 13, 14] that use a protonation coordinate λ to distinguish the protonated from the deproto-

nated state. Here, we follow and expand the λ-dynamics approach of Brooks et al. [27] and treat λ as an additional degree of freedom in the Hamiltonian with mass m_λ. Each protonatable group is associated with its own λ "particle" that adopts continuous values in the interval $[0, 1]$, where the end points around $\lambda = 0$ and $\lambda = 1$ correspond to the physical protonated or deprotonated states. A barrier potential with its maximum at $\lambda \approx 0.5$ serves two purposes. (i) It reduces the time spent in unphysical states, and (ii) it allows to tune for optimal sampling of the λ coordinate by adjusting its height [8, 9]. Current λ-dynamics simulations with GROMACS are however limited to small system sizes with a small number n_λ of protonatable groups [8, 7, 9], as the existing, PME-based implementation (see `www.mpibpc.mpg.de/grubmueller/constpH`) needs an extra PME mesh evaluation per λ group and suffers from the PME parallelization problem. While these extra PME evaluations can be overcome for the case where only the charges differ between the states, for the most general case of chemical alterations this is not possible.

Without the PME parallelization limitations, a significantly higher number of compute nodes could be utilized, so that both larger and more realistic biomolecular systems would become accessible. The Fast Multipole Method [15] (FMM) is a method that by construction parallelizes much better than PME. Beyond that, the FMM can compute and communicate the additional multipole expansions that are required for the local charge alternatives of λ groups with far less overhead as compared to the PME case. This makes the communicated volume (extra multipole components) somewhat larger, but no global communication steps are involved as in PME, where the global communication volume grows linearly with n_λ and quadratic with p. We also considered other methods that, like FMM, scale linearly with the number of charges, as e.g. multigrid methods. We decided in favor of FMM, because it showed better energy conservation and higher performance in a comparison study [2].

We will now introduce λ-dynamics methods and related work to motivate the special requirements they have on the electrostatics solver. Then follows an overview of our FMM-based solver and the design decisions reflecting the specific needs of MD simulation. We will describe several of the algorithmical and hardware-exploiting features of the implementation such as error control, automatic performance tuning, the lightweight tasking engine, and the CUDA-based GPU implementation.

2 Chemical variability and protonation dynamics

Classical MD simulations employ a Hamiltonian $\mathcal{H}$ that includes potential terms modeling the bonded interactions between pairs of atoms, the bond angle interactions between bonded atoms, and the van der Waals and Coulomb interactions between all pairs of atoms. For conventional, force field based MD simulations, the chemistry of molecules is fixed during a simulation because chemical changes are not described by established biomolecular force fields. Exceptions are alchemical trans-

formations [47, 36, 46, 38], where the system is either driven from a state A described by Hamiltonian $\mathcal{H}_\mathrm{A}$ to a slightly different state B (with $\mathcal{H}_\mathrm{B}$) via a λ parameter that increases linearly with time, or where A/B chimeric states are simulated at several fixed λ values between $\lambda = 0$ and $\lambda = 1$, as e.g. in thermodynamic integration [24]. The A $\rightarrow$ B transition is described by a combined, λ-dependent Hamiltonian

$$\mathcal{H}_\mathrm{AB}(\lambda) = (1-\lambda)\mathcal{H}_\mathrm{A} + \lambda\mathcal{H}_\mathrm{B}. \tag{1}$$

In these simulations, which usually aim at determining the free energy difference between the A and B states, the value of λ is an input parameter.

In contrast, with λ-dynamics [27, 25, 16], the λ parameter is treated as an additional degree of freedom with mass m, whose 1D coordinate λ and velocity $\dot{\lambda}$ evolve dynamically during the simulation. Whereas in a normal MD simulation all protonation states are fixed, with λ-dynamics, the pH value is fixed instead and the protonation state of a titratable group changes back and forth during the simulation in response to its local electrostatic environment [39, 23]. If two states (or *forms*) A and B are involved in the chemical transition, the corresponding Hamiltonian expands to

$$\mathcal{H}(\lambda) = (1-\lambda)\mathcal{H}_\mathrm{A} + \lambda\mathcal{H}_\mathrm{B} + \frac{m}{2\dot{\lambda}^2} + V_\mathrm{bias}(\lambda) \tag{2}$$

with a bias potential V_bias that is calibrated to reflect the (experimentally determined) free energy difference between the A and B states and that optionally controls other properties relating to the A $\leftrightharpoons$ B transitions [8]. With the potential energy part V of the Hamiltonian, the force acting on the λ particle is

$$f_\lambda = -\frac{\partial V}{\partial \lambda}. \tag{3}$$

If coupled to the protonated and deprotonated form of an amino acid side chain, e.g., λ-dynamics enables dynamic protonation and deprotonation of this side chain in the simulation (see Fig. 1 for an example), accurately reacting to the electrostatic environment of the side chain. More generally, also alchemical transformations beyond protons are possible, as well as transformations involving more than just two forms A and B. Equation 2 shows the Hamiltonian for the simplest case of a single protonatable group with two forms A and B, but we have extended the framework to multiple protonatable groups using one λ_i parameter for each chemical form [8, 9, 7].

2.1 Variants of λ-dynamics and the bias potential

The key aim of λ-dynamics methods is to allow for dynamic protonation, but there are three areas in which the implementations differ from each other. These are the coordinate system used for λ, the type of the applied bias potential, and how λ is coupled to the alchemical transition. Before we discuss the different choices, let us define two terms used in the context of chemical variability and protonation. We

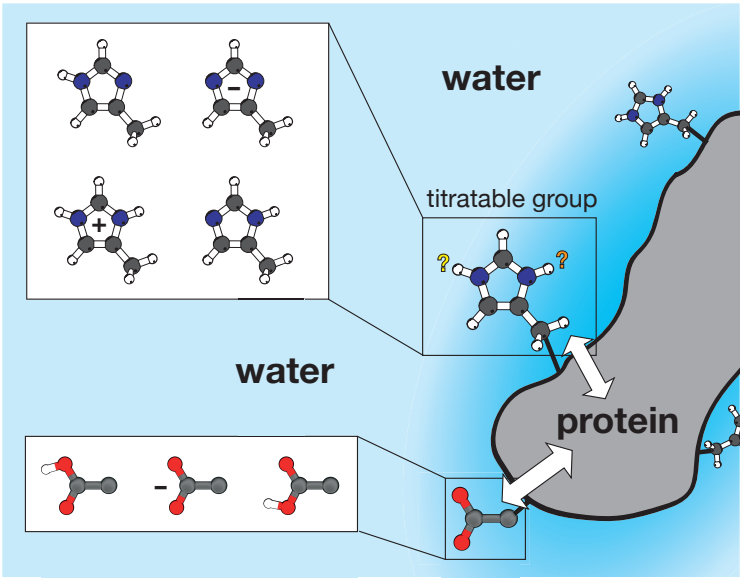


Figure 1: Simplified sketch of a protein (right, grey) in solution (blue) with several protonatable sites (ball-and-stick representations) of which a histidine (top left) and a carboxyl group (bottom left) are highlighted. The histidine site contains four forms (two neutral, two charged), whereas the carboxyl group contains three forms (two neutral, one negatively charged). In λ-dynamics, the lambdas controls how much of each form is contributing to a site. Atom color coding: carbons-black, hydrogens/protons-white, oxygens-red, nitrogens-blue.

Table 1: Three variants of λ-dynamics are considered.

variant name	dynamical variable	geometric picture
linear [9]	λ	λ lives on a constricted linear interval, e.g. [0,1]
hypersphere [8]	θ	λ lives on a circle
Brooks' Nexp [26]	ϑ	no simple geometric interpretation

use the term **site** for a part of a molecule that can interconvert between two or more chemically different states, e.g. the protonated and deprotonated forms of an aminoacid. Additionally, we call each of the chemically different states of a site a **form**. For instance, a protonatable group is a site with at least two forms A and B, a protonated form A and a deprotonated form B.

The coordinate system for λ

Based on the coordinate system in which λ lives (or on the dynamical variables used to express λ), we consider three variants of λ-dynamics listed in Tab. 1. The *linear* variant is conceptually most straightforward, but it definitely needs a bias potential to constrain λ to the interval [0..1]. The circular coordinate system for λ used in the *hypersphere* variant automatically constrains the range of λ values to the desired interval, however one needs to properly correct for the associated circle entropy [8]. The *Nexp* variant implicitly fulfils the constraints on the N_{forms} individual lambdas (Equation 4) for sites that are allowed to transition between N_{forms} different forms ($N_{\text{forms}} = 2$ in the case of simple protonation), such that no additional constraint solver for the λ_i is needed.

The bias potential

The bias potential $V_{\text{bias}}(\lambda)$ that acts on λ fulfils one or more of the following tasks.

1. If needed, it limits the accessible values of λ to the interval [0..1], whereas slight fluctuations outside that interval may be desirable (Fig. 2A).

2. It cancels out any unwanted barrier at intermediate λ values (B)

3. It takes care that the resulting λ values cluster around 0 or 1, suppressing values between about 0.2 and 0.8 (C)

4. It regulates the depth and width of the minima at 0 and 1, such that the resulting λ distribution fits the experimental free energy difference between protonated and deprotonated form (C+D).

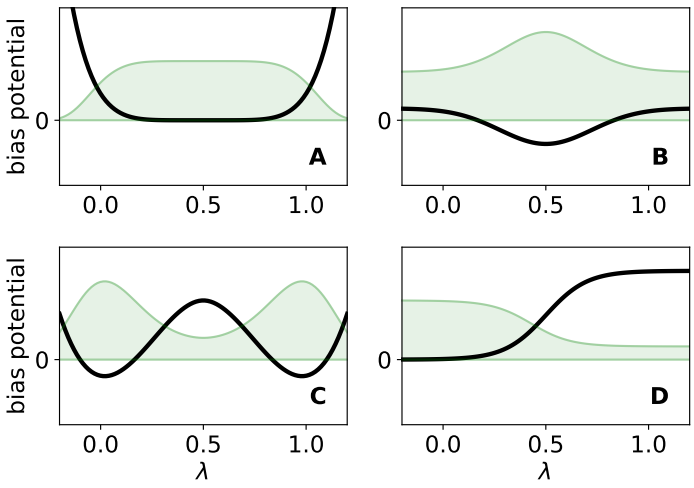


Figure 2: Qualitative sketches of individual bias potentials (black) that fulfil some of the requirements (i)–(v), and resulting equilibrium distributions of λ values (green).

5. It allows to tune for optimal sampling of the λ space by adjusting the barrier height at $\lambda = 0.5$ (C)

Taken together, the various contributions to the barrier potential might look like the example given in Fig. 3 for a particular λ in a simulation.

How λ controls the transition between states

The λ parameter can either be coupled to the *transition* itself between two forms (as in [8, 9]), then $\lambda = 0$ corresponds to form A and $\lambda = 1$ to form B. Alternatively, each form gets assigned its own λ_α with $\alpha \in \{A, B\}$ as *weight* parameter. In the latter case one needs extra constraints on the weights similar to

$$\sum \lambda_\alpha = 1, \qquad 0 \leq \lambda_\alpha \leq 1, \tag{4}$$

such that only one of the physical forms A or B is fully present at a time. For the examples mentioned so far, with just two forms, both approaches are equivalent and one would rather choose the first one, because it involves only one λ and needs no extra constraints.

If, however, a site can adopt more than two chemically different forms, the weight approach can become more convenient as it allows to treat sites with any number N_{forms} of forms (using a number of N_{forms} independent λ parameters). Further, it does not require that the number of forms is a power of two ($N_{\text{forms}} = 2^{N_\lambda}$) as in the transition approach.

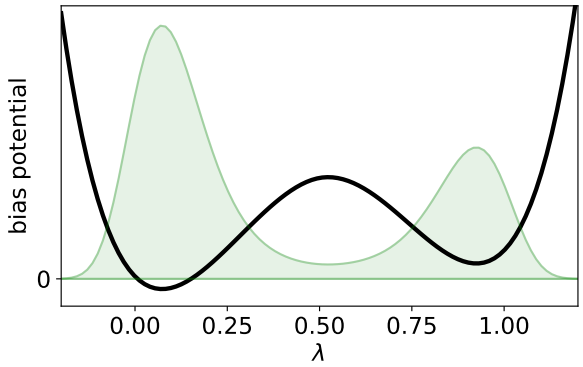


Figure 3: Qualitative sketch of a bias potential (black) that fulfils all requirements (i)–(v) with resulting equilibrium distribution of λ values (green).

2.2 Keeping the system neutral with buffer sites

In periodic boundary conditions as typically used in MD simulations, the electrostatic energy is only defined for systems with a zero net charge. Therefore, if the charge of the MD system changes due to λ mediated (de)protonation events, system neutrality has to be preserved. With PME, any net charge can be artificially removed by setting the respective Fourier mode's coefficient to zero, so that also in these cases a value for the electrostatic energy can be computed. However, it is merely the energy of a similar system with a neutralizing background charge added. Severe simulation artifacts have been reported as side effects of this approach [19].

As an alternative, a charge buffer can be used that balances the net charge of the simulation system arising from fluctuating charge of the protonatable sites [48, 9]. A reduced number of n_{buffer} buffer sites, each with a fractional charge $|q| \leq 1e$ (e.g. via $H_2O \rightleftharpoons H_3O^+$), was found to be sufficient to neutralize the N_{sites} protonatable groups of a protein with $n_{\text{buffer}} \ll N_{\text{sites}}$. The total charge of these buffer ions is coupled to the system's net charge with a holonomic constraint [9]. The buffer sites should be placed sufficiently far from each other, such that their direct electrostatic interaction through the shielding solvent is negligible.

3 A modern FMM implementation in C++ tailored to MD simulation

High performance computing (HPC) biomolecular simulations differ from other scientific applications by their comparatively small particle numbers and by their extremely high iteration rates. With GROMACS, when approaching the scaling limit, the number of particles per CPU core typically lies in the order of a few hundred, whereas the wall-clock time required for computing one time step lies in

the range of a millisecond or less [42]. In MD simulations with λ-dynamics, the additional challenge arises to calculate the energy and forces from a Hamiltonian similar to Equation 2, but for N protonatable sites, in an efficient way. In addition to the Coulomb forces on the regular charged particles, the electrostatic solver has to compute the forces on the N λ particles as well [8] via

$$\begin{aligned} f_{\lambda_i} = -\frac{\partial V_C}{\partial \lambda_i} &= -\frac{\partial V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i, \lambda_{i+1}, \ldots \lambda_N)}{\partial \lambda_i} \\ &= -\Big[V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i = 1, \lambda_{i+1}, \ldots \lambda_N) \\ &\quad -V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i = 0, \lambda_{i+1}, \ldots \lambda_N)\Big] \end{aligned} \tag{5}$$

Accordingly, with λ-dynamics, for each of the λ_i's, the energies of the pure (i.e., $\lambda_i = 0$ and $\lambda_i = 1$) states have to be evaluated while keeping all other lambdas at their actual fractional values.

The aforementioned requirements of biomolecular electrostatics have driven several design decisions in our C++ FMM, which is a completely new C++ reimplementation of the Fortran ScaFaCoS FMM [5]. Although several other FMM implementations exist [50, 1], none of them is prepared to compute the potential terms needed for biomolecular simulations with λ-dynamics.

Although our FMM is tailored for usage with GROMACS, it can be used as an electrostatics solver for other applications as well as it comes as a separate library in a distinct Git repository. On the GROMACS side we provide the necessary modifications such that FMM instead of PME can be chosen at run time. Apart from that, GROMACS calls our FMM library via an interface that can also be used by other codes. The development of this library follows three principles. First, the building blocks (i.e., data structures) used in the FMM support each level of the hierarchical parallelism available on today's hardware. Second, the library provides different implementations of the involved FMM operators depending on the underlying hardware. Third, the library optionally supports λ-dynamics via an additional interface.

3.1 The FMM in a nutshell

The FMM approximates and thereby speeds up the computation of the Coulomb potential V_C for a system of N charges:

$$V_\mathrm{C} \propto \sum_{i}^{N} \sum_{j<i} \frac{q_i q_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{6}$$

For that purpose, the FMM divides the simulation box into eight smaller boxes (depth $d = 1$), which are subsequently subdivided into eight smaller boxes again

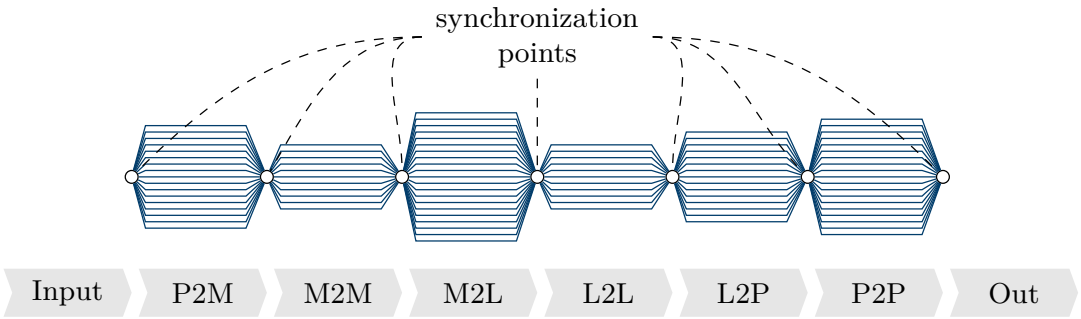


Figure 4: The classical (sequential) FMM workflow consists of six stages. Only the nearfield (P2P) can be computed completely independent of all other stages. Each farfield stage (P2M, M2M, etc.) depends on the former stage and exhibits different amounts of parallelism. Especially the distribution of multipole and local moments in the tree provide limited parallelism in classical loop-based parallelization schemes.

($d = 2$) and again ($d = 3, 4, \ldots$). The depth d refers to the number of subdivisions. On the lowermost level, i.e. for the smallest boxes (largest d), all interactions between neighboring boxes are directly calculated (these are called the near-field interactions). Interactions with boxes further away are approximated by a multipole expansion of order p (these are called the far-field interactions). A comprehensive description of the FMM algorithm is beyond the scope of this text, however we will shortly describe the basic workflow and the different operators used in the six FMM stages as these will be referred to in the following sections. For a detailed overview of the FMM, see [22]; for an introduction in our C++ FMM implementation see [12].

FMM workflow

The FMM algorithm consists of six different stages, five of them required for the farfield (FF) and one for the nearfield (NF) (Fig. 4). After setting up the FMM parameters tree depth (d) and multipole order (p), the following workflow is executed.

1. **P2M**: Expand particles into spherical multipole moments ω_{lm} up to order p on the lowest level for each box in the FMM tree. Multipole moments for particles in the same box can be summed into a multipole expansion representing the whole box.

2. **M2M**: Translate the multipole expansion of each box to its parent box inside the tree. Again, multipole expansions with the same box center can be summed up. The translation up the tree is repeated until the root node is reached.

3. **M2L**: Transform remote multipole moments ω_{lm} into local moments μ_{lm} for each box on every level. Only a limited number of interactions for each box on each level is performed to achieve linear scaling.

4. **L2L**: Translate local moments μ_{lm} starting from the root node down towards the leaf nodes. Local moments within the same box are summed.

5. **L2P**: After reaching the leaf nodes, the farfield contributions for the potentials Φ_{FF}, forces $\mathbf{F}_{\mathrm{FF}}$, and energy E_{FF} are computed.

6. **P2P**: Interactions between particles within each box and its direct neighbors are computed directly, resulting in the nearfield contributions for the potentials Φ_{NF}, forces $\mathbf{F}_{\mathrm{NF}}$, and energy E_{NF}.

Features of our FMM implementation

Our FMM implementation includes special algorithmical features and features that help to optimally exploit the underlying hardware. Algorithmical features are

- Support for open and 1D, 2D and 3D periodic boundary conditions for cubic boxes.
- Support for λ-dynamics (Sec. 2).
- Communication-avoiding algorithms for internode communication via MPI (Fig. 9).
- Automatic tuning of FMM parameters d and p to provide automatic error control and runtime minimization [6] based on a user-provided energy error threshold ΔE (Fig. 10).
- Adjustable tuning to reduce or avoid energy drift (Fig. 11).

Hardware features include

- A performance-portable SIMD layer (Sec. 3.2.1).
- A light-weight, NUMA-aware task scheduler for CPU and GPU tasks (Sec. 3.2.2).
- A GPU implementation based on CUDA (Sec. 3.4).

3.2 Utilizing hierarchical parallelism

3.2.1 Intra-core parallelism

A large fraction of today's HPC peak performance stems from the increasing width of SIMD vector units. However, even modern compilers cannot generate fully vectorized code unless the data structures and dependencies are very simple. Generic algorithms like FFTs or basic linear algebra can be accelerated by using third-

party libraries and tools specifically tuned and optimized for a multitude of different hardware configurations. Unfortunately, the FMM data structures are not trivially vectorizable and require careful design. Therefore, we developed a performance-portable SIMD layer for non-standard data structures and dependencies in C++.

Using only C++11 language features without third-party libraries allows to fine-tune the abstraction layer for the non-trivial data structures and achieve a better utilization. Compile-time loop-unrolling and tunable stacking are used to increase out-of-order execution and instruction-level parallelism. Such optimizations depend heavily on the targeted hardware and must not be part of the algorithmic layer of the code. Therefore, the SIMD layer serves as an abstraction layer that hides such hardware-specifics and that helps to increase code readability and maintainability. The requested SIMD width (1x, 2x, ..., 16x) and type (float, double) is selected at compile time. The overhead costs and performance results are shown in Fig. 5. The baseline plot (blue) shows the costs of the M2L operation (float) without any vectorization enabled. All other plots show the costs of the M2L operation (float) and 16-fold vectorization (AVX-512). Since the runtime of the M2L operation is limited by the loads of the M2L operator, we try to amortize these costs by utilizing multiple ($2\times \ldots 6\times$) SIMDized multipole coefficient matrices together with a single operator via unrolling (stacking). As can be seen in Fig. 5, unrolling the multipole coefficient matrices $2\times$ (red), we reach the minimal computation time and the expected 16-fold speedup. Additional unroll factors ($3\times \ldots 6\times$) will not improve performance due to register spilling. To reach optimal performance, it is required to reuse (cache) the M2L operator for around 300 (or more) of these steps.

3.2.2 Intra-node and inter-node parallelism

To overcome scaling bottlenecks of a pragma-based loop-level parallelization (see Fig. 4), our FMM employs a lightweight tasking framework purely based on C++. Being independent of other third-party tasking libraries and compiler extensions allows to utilize resources better, since algorithm-specific behavior and data-flow can be taken into account. Two distinct design features are a type-driven priority scheduler and a static dataflow dispatcher. The scheduler is capable of prioritizing tasks depending on their type at compile time. Hence, it is possible to prioritize vertical operations (like M2M and L2L) in the tree. This reduces the runtime twofold. First, it reduces the scheduling overhead at runtime by avoiding costly virtual function calls. Second, since the execution of the critical path is prioritized, the scheduler ensures that a sufficient amount of independent parallelism gets generated. The dataflow dispatcher defines the dependencies between tasks – a data flow graph – also at compile time (see Fig. 6). Together with loadbalancing and workstealing strategies, even a non-trivial FMM data flow can be executed. For compute-bound problems this design shows virtually no overhead. However, in MD we are interested in smaller particle systems with only a few hundred particles per compute node. Hence, we have to take even more hardware constraints into account. Performance penalties due to the memory hierarchy (NUMA) and costs to access

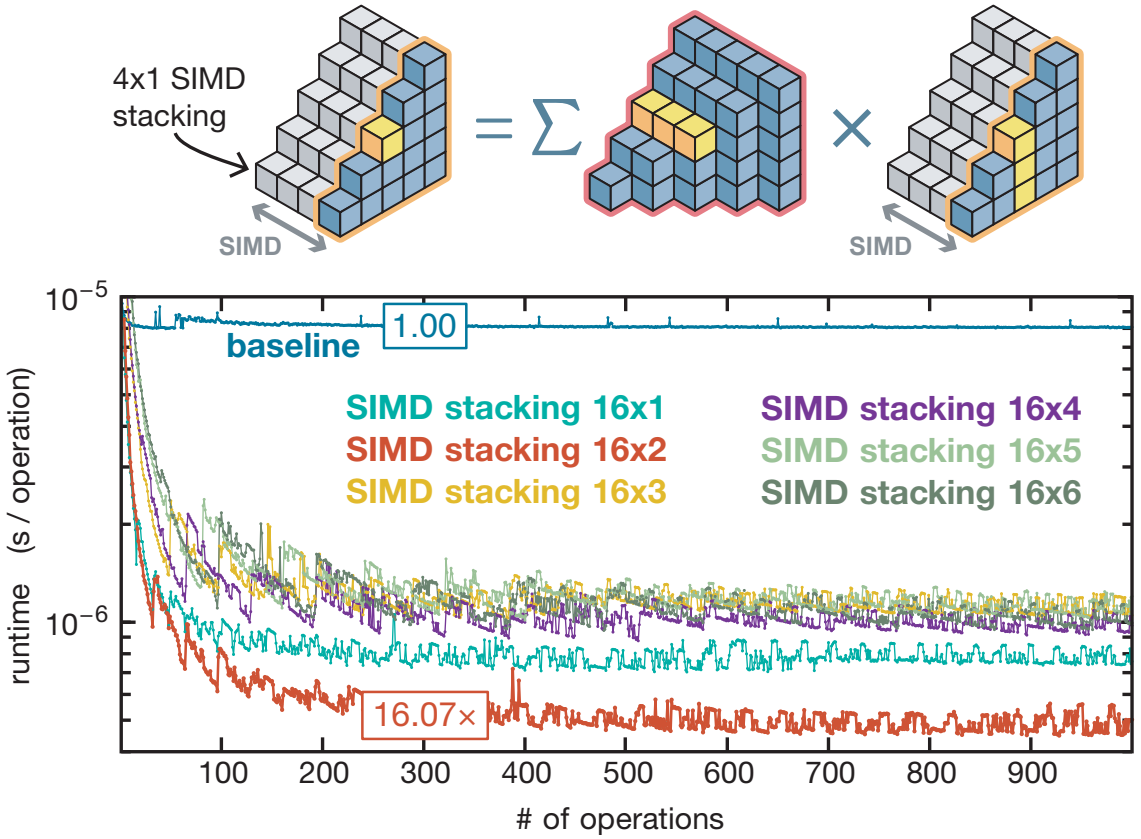


Figure 5: M2L operation benchmark for vectorized data structures with multipole order $p = 10$ on an Intel Xeon Phi 7250F CPU for a float type with 16x SIMD (AVX-512). The benchmarks shows the performance of different SIMD/unrolling combinations. E.g. the red curve (SIMD stacking 16×2) utilizes 16-fold vectorization together with 2-fold unrolling. For a sufficient number(around 300) of vectorized operations, a 16-fold improvement can be measured for the re-designed data structures.

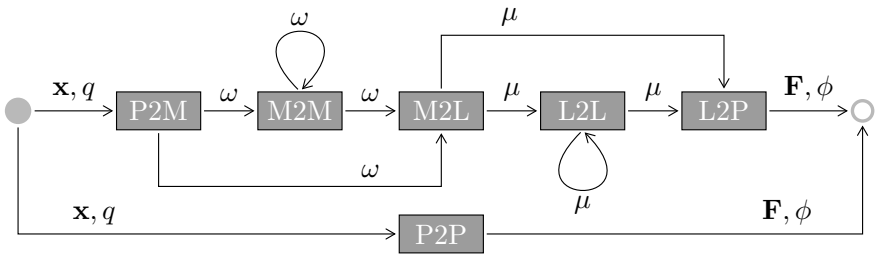


Figure 6: The data flow of the FMM still consists of six stages. However, synchronization now happens on a fine-grained level and not only after each full stage is completed. This allows to overlap parts that exhibit poor parallelization with parts that show a high degree of parallel code. The dependencies of such a data flow graph can be evaluated and even prioritized at compile time.

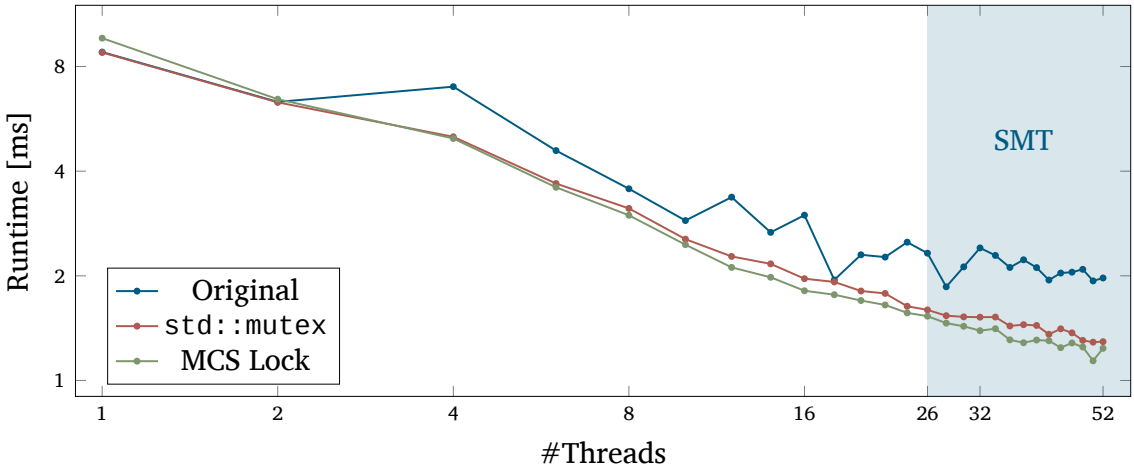


Figure 7: Intranode FMM benchmark for 1,000 particles, multipole order $p = 1$ and tree depth $d = 3$ on a 2x26-core Intel Xeon Platinum 8170 CPU. When using MCS locks, simultaneous multithreading and 50 threads, the overall improvement compared to the original implementation reaches $> 40\%$, translating into a reduction in runtime from 1.93 ms down to 1.14 ms.

memory in a shared fashion via locks introduce additional overhead. Therefore, we extended also our tasking framework with NUMA-aware memory allocations, workstealing and scalable Mellor-Crummey Scott (MCS) locks [35] to enhance the parallel scalability over many threads, as shown in Fig. 7.

In the future, we will extend our tasking framework so that tasks can also be offloaded to local accelerators like GPUs, if available on the node.

For the node-to-node communication via MPI the aforementioned concepts do not work well (see Fig. 8), since loadbalancing or workstealing would create large overheads due to a large amount of small messages. To avoid or reduce the latency that comes with each message, we employ a communication-avoiding parallelization scheme [10]. Nodes do not communicate separately with each other, but form groups in order to reduce the total number of messages. At the same time the message size can be increased. Depending on the total number of nodes involved, the group size parameter can be tuned for performance (see Fig. 9).

3.3 Algorithmic interface

Choosing the optimal FMM parameters in terms of accuracy and performance is difficult if not impossible to do manually as they also depend on the charge distribution itself. A naive choice of tree depth d and multipole order p might either lead to wasting FLOPs or to results that are not accurate enough. Therefore, d and p are automatically tuned depending on the underlying hardware and on a provided energy tolerance ΔE (absolute or relative acceptable error in Coulombic energy). The corresponding parameter set $\{d, p\}$ is computed such that the accuracy is met

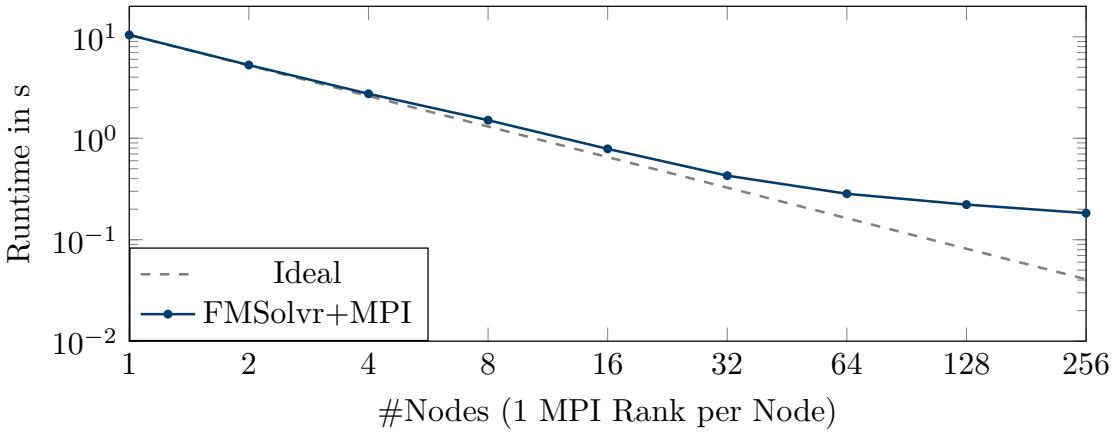


Figure 8: Initial internode FMM benchmark for 1,000,000 particles, multipole order $p = 3$ and tree depth $d = 5$ with one MPI rank per compute node of the JURECA cluster.

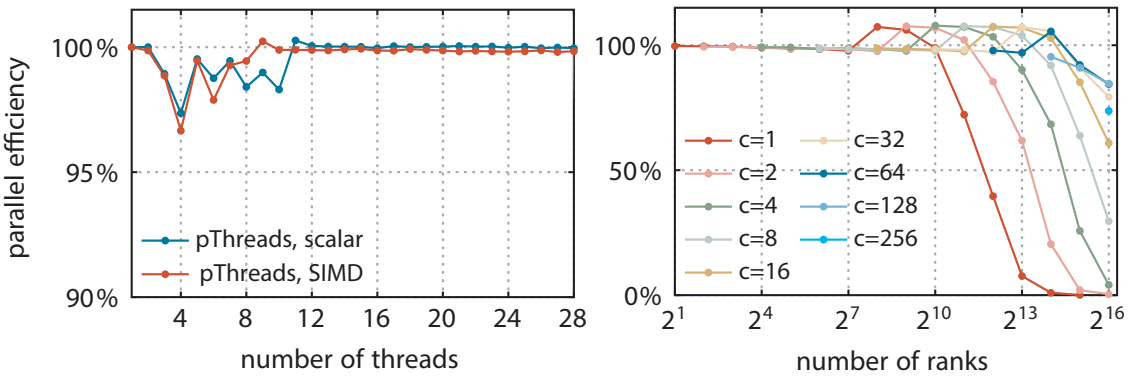


Figure 9: **Left:** Intranode FMM parallelization — efficiency of different threading implementations. Near field interaction of 114,537 particles in double precision on up to 28 cores on a single node with two 14-core Intel Xeon E5-2695 v3 CPUs. Single precision computation as well as other threading schemes (std::thread, boost::thread, OpenMP) showed similar excellent scaling behavior. The plot has been normalized to the maximum turbo mode frequency which varies with the number of active cores (3.3–2.8 GHz for scalar operation, 3.0–2.6 GHz for SIMD operation). **Right:** Internode parallelization — strong scaling efficiency of a communication avoiding, replication-based workload distribution scheme [10]. Near field interaction of 114,537 particles on up to 65,536 Blue Gene/Q cores using replication factor c. In the initial replication phase, only c nodes within a group communicate. Afterwards, communication is restricted to all pairs of p/c groups. For 65,536 cores, i.e. only 1–2 particles per core initially, a maximum parallel efficiency of 84 % (22 ms runtime) is reached for $c = 64$, and the maximal replication factor $c = 256$ yields an efficiency of 73 %, while a classical particle distribution ($c = 1$) would require a runtime exceeding 1 minute due to communication latency.

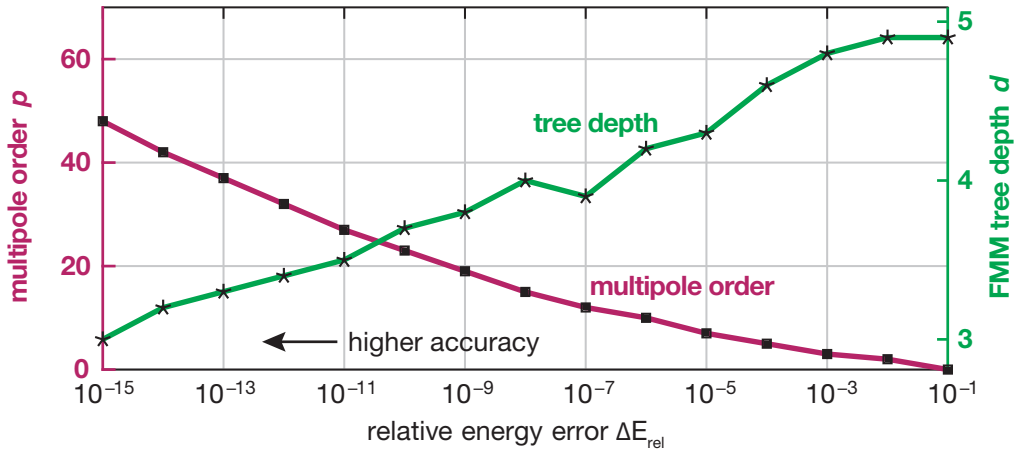


Figure 10: Depending on a maximum relative or absolute energy tolerance ΔE, the automatic runtime minimization provides the optimal set of FMM input parameters $\{d, p\}$. A lower requested error in energy results in an increased multipole order p (magenta). Since the computational complexity of the farfield operators M2M, M2L and L2L scales with p^3 or even p^4 (depending on the used implementation), the tree depth d is reduced accordingly to achieve a minimal runtime (green). With fractional depths [49], as used here, the runtime can be optimized even more than with integer depths.

at minimal computational costs (Fig. 10) [6].

Besides tuning the accuracy to achieve a certain acceptable error in the Coulombic energy for each time step, the FMM can additionally be tuned to reduce the energy drift over time.

Whereas multipole orders of about ten yield a comparable drift of the total energy over time as a typical simulation with PME, the drift with FMM can be reduced to much lower levels if desired (Fig. 11).

3.4 CUDA implementation of the FMM for GPUs

A growing number of HPC clusters incorporate accelerators like GPUs to deliver a large part of the FLOPS. Also GROMACS evolves towards offloading more and more tasks to the GPU, for reasons of both performance and cost-efficiency [32, 31].

For system sizes that are typical for biomolecular simulations, FMM performance critically depends on the M2L and P2P operators. For multipole orders of about eight and larger their execution times dominate the overall FMM runtime (Fig. 12).

Hence, these operators need to be parallelized very efficiently on the GPU. At the same time, all remaining operators need to be implemented on the GPU as well to

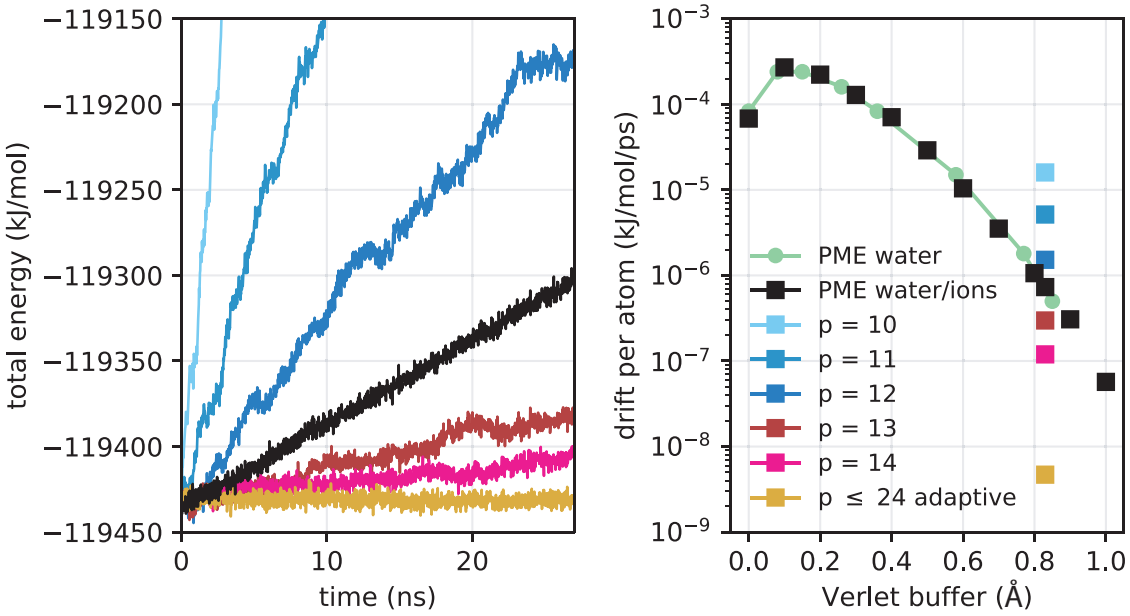


Figure 11: Observed drift of the total energy for different electrostatics settings. **Left:** evolution of the total energy for PME with order 4, mesh distance 0.113 nm, `ewald-rtol` set to 10^{-5} (black line) as well as for FMM with different multipole orders p at depth $d = 3$ (see legend in the right panel). Test system is a double precision simulation at $T \approx 300$ K in periodic boundaries of 40 Na^+ and 40 Cl^- ions solvated in a 4.07 nm^3 box containing extended simple point charge (SPC/E) water molecules [3], comprising 6,740 atoms altogether. Time step $\Delta t = 2$ fs, cutoffs at 0.9 nm, pair-list updated every 10 steps. **Right:** Black squares show the drift with PME for different Verlet buffer sizes for the water/ions system using 4×4 cluster pair lists [41]. For comparison, green line shows the same for pure SPC/E water (without ions) taken from Ref. [34]. Influence of different multipole orders p on the drift is shown for a fixed buffer size of 8.3 Å. The GROMACS default Verlet buffer settings yield a drift of $\approx 8 \times 10^{-5}$ kJ/mol/ps per atom for these MD systems, corresponding to the first data point on the left (black square / green circle).

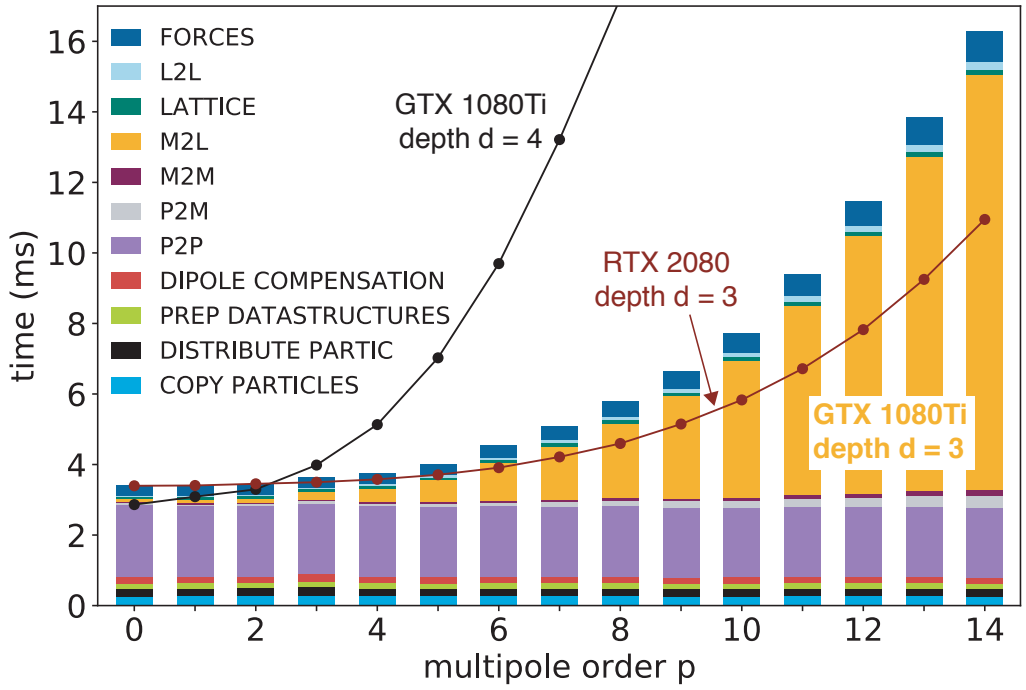


Figure 12: Colored bars show detailed timings for the various parts of a single FMM step on a GTX 1080Ti GPU for a 103,000 particle system using depth $d = 3$. For comparison, total execution time for $d = 3$ on an RTX 2080 GPU is shown as brown line, whereas black line shows timings for $d = 4$ on a GTX 1080Ti GPU. CUDA parallelization is used in each FMM stage leaving the CPU mostly idle.

avoid memory traffic between device (GPU) and host (CPU) that would otherwise become necessary. This traffic would introduce a substantial overhead as a complete MD time step may take just a few milliseconds to execute.

Our encapsulated GPU FMM implementation takes particle positions and charges as input and returns the electrostatic forces on the particles as output. Memory transfers between host and device are performed only at these two points in the calculation step.

The particle positions and charges are split into different CUDA streams that allow for asynchronous memory transfer to the host. The memory transfer is overlapped with the computation of the spatial affiliation of the octree box.

In contrast to the CPU FMM that utilizes $\mathcal{O}(p^3)$ far field operators (M2M, M2L, L2L), the GPU version is based on the $\mathcal{O}(p^4)$ operator variant. The $\mathcal{O}(p^3)$ operators require less multiplications to calculate the result, but they introduce additional highly irregular data structures to rotate the moments. Since the performance of the GPU FMM at small multipole orders is not limited by the number of floating point operations (Fig. 12) but rather by scattered memory access patterns, we use the $\mathcal{O}(p^4)$ operators for the GPU implementation.

We will now outline our CUDA implementation of the operations needed in the various stages of the FMM (Figs. 4–6), which starts by building the multipoles on the lowest level with the P2M operator.

P2M – Particle To Multipole

The P2M operation is described in detail elsewhere [44]. The large number of registers that is required and the recursive nature of this stage limits the efficient GPU parallelization. The operation is however executed independently for each particle and the requested multipole expansion is gained by summing atomically into common expansion points. The result is precomputed locally using shared memory or intra-warp communication to reduce the global memory traffic when storing the multipole moments. The multipole moments ω, local moments μ and the far field operators $\mathbf{A}, \mathbf{M}$, and $\mathbf{C}$ are stored as triangular shaped matrices

$$\omega, \mu, \mathbf{A}, \mathbf{C} \in \mathbb{K}^{p \times p} := \{(x_{lm})_{l=0,\dots,p,\ m=-l,\dots,l} \mid x_{lm} \ \in \mathbb{C} \ \} \tag{7}$$

and $\mathbf{M} \in \mathbb{K}^{2p \times 2p}$, where p is the multipole order.

To map the triangular matrices efficiently to contiguous memory, their elements are stored as 1D arrays of complex values and the l, m indices are calculated on the fly when accessing the data. For optimal performance, different stages of the FMM require different memory access patterns. Therefore, the data structures are stored redundantly in a Structure of Arrays (SoA) and Array of Structures (AoS) version. The P2M operator writes to AoS, whereas the far field operators use SoA. A copy

kernel, negligible in runtime, does the copying from one structure to another.

M2M – Multipole To Multipole

The M2M operation, which shifts the multipole expansions of the child boxes to their parents, is executed on all boxes within the tree, except for the root node which has no parent box. The complexity of this operation is $\mathcal{O}(p^4)$; one M2M operation has the form

$$\omega_{lm}(a') = \sum_{j=0}^{l} \sum_{k=-j}^{j} \omega_{jk}(a) \mathbf{A}_{l-j,m-k}(a - a'), \tag{8}$$

where $\mathbf{A}$ is the M2M operator and a and a' are different expansion center vectors. The operation performs $\mathcal{O}(p^2)$ dot products between ω and a part of the operator $\mathbf{A}$. These operations need to be executed in all boxes in the octree, excluding the box on level 0, i.e. the root node. The kernels are executed level wise on each depth, synchronizing between each level. Each computation of the target ω_{lm} for a distinct (l, m) pair is performed in a different CUDA block of the kernel, with threads within a block accessing different boxes sharing the same operator. The operator can be efficiently preloaded into CUDA shared memory and is accessed for different ω_{lm} residing in different octree boxes. Each single reduction step is performed sequentially by each thread. This has the advantage that the partial products are stored locally in registers, reducing the global memory traffic since only $\mathcal{O}(p^2)$ elements are written to global memory. It also reduces the atomic accesses, since the results from eight distinct multipoles are written into one common target multipole.

M2L – Multipole To Local

The M2L operator works similarly to M2M, but it requires much more transformations as each source ω is transformed to 189 target μ boxes. The group of boxes to which a particular ω is transformed to is called the interaction set. It contains all child boxes of the direct neighbor boxes of the source's ω parent. The M2L operation is defined as

$$\mu_{lm}(r) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \omega_{jk}(a) \mathbf{M}_{l+j,m+k}(a - r), \tag{9}$$

where r and a are different expansion centers. The operation differs only slightly from M2M in the access pattern but is of the same $\mathcal{O}(p^4)$ complexity. As the M2L runtime is crucial for the overall FMM performance, we have implemented several parallelization schemes. Which scheme is the fastest depends on tree depth and multipole order. The most efficient implementation is based on presorted lists con-

taining interaction box pointers. The lists are presorted so that the symmetry of the operator $\mathbf{M}$ can be exploited. In $\mathbf{M}$, the orthogonal operator elements differ only by their sign. Harnessing this minimizes the number of multiplications and global memory accesses and allows to reduce the number of spawned CUDA blocks from 189 to 54. However, it introduces additional overhead in logic to change signs and computations of additional target μ box positions, so the performance speedup is smaller than 189/54. The kernel is spawned similarly to the M2M kernel performing one dot product per CUDA block preloading the operator $\mathbf{M}$ into shared memory. The sign changing is done with the help of and additional bitset provided for each operator. Three different parallelization approaches are compared in Fig. 13. Considering the hardware performance bottlenecks of this stage, the limitations highly differ for particular implementations. The naive M2L kernel is clearly bandwidth limited and achieves nearly 500 GB/s for multipole orders larger than ten. This is higher than the theoretical memory throughput of the tested GPU, which is 480 GB/s, due to caching effects. The cache utilization is nearly at 100% achieving 3500 Gb/s. However, the performance of this kernel can be enhanced further by moving towards more compute bound regime. With the dynamical approach the performance is mainly limited by the costs of spawning additional kernels. It can be clearly seen with the flat curve shape for multipoles smaller than 13 in Fig. 13. The hardware utilization for the symmetrical kernel is depicted in Fig. 14. The performance of this kernel depends on the multipole order p, since p^2 is a CUDA gridsize parameter [40]. The values $p < 7$ lead to underutilization of the underlying hardware, however they are mostly not of practical relevance. For larger values the performance is operations bound achieving about 80% of the possible compute utilization.

L2L – Local To Local

The L2L operation is executed for each box in the octree, shifting the local moments from the root of the tree down to the leaves, opposite in direction to M2M. Although the implementation is nearly identical, it achieves slightly better performance than M2M because the number of atomic memory accesses is reduced due to the tree traversing direction. For the L2L operator, the result is written into eight target boxes, whereas M2M gathers information from eight source boxes into one.

L2P – Local To Particles

The calculation of the potentials at particle positions x_i requires evaluating

$$\Phi(x_i) = \sum_{l=0}^{p} \sum_{m=-l}^{l} \mu_{lm} \mathring{\omega}_{lm}^{i}, \quad i = 0, ..., N_{\text{box}}\,, \tag{10}$$

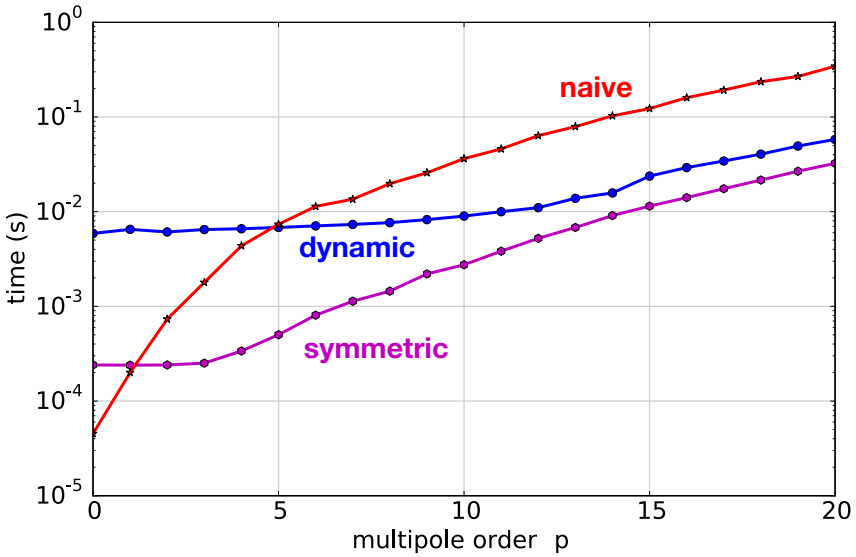


Figure 13: Comparison of three different parallelization schemes for the M2L operator, which is the most compute intensive part of the FMM algorithm. The naive implementation (red) directly maps the operator loops to CUDA blocks. It beats the other schemes only for orders $p < 2$. Dynamic parallelization (blue) is a CUDA specific approach that dynamically spawns thread groups from the kernels. The symmetric scheme (magenta) represents the FMM tree via presorted interaction lists. It also exploits the symmetry of the M2L operator.

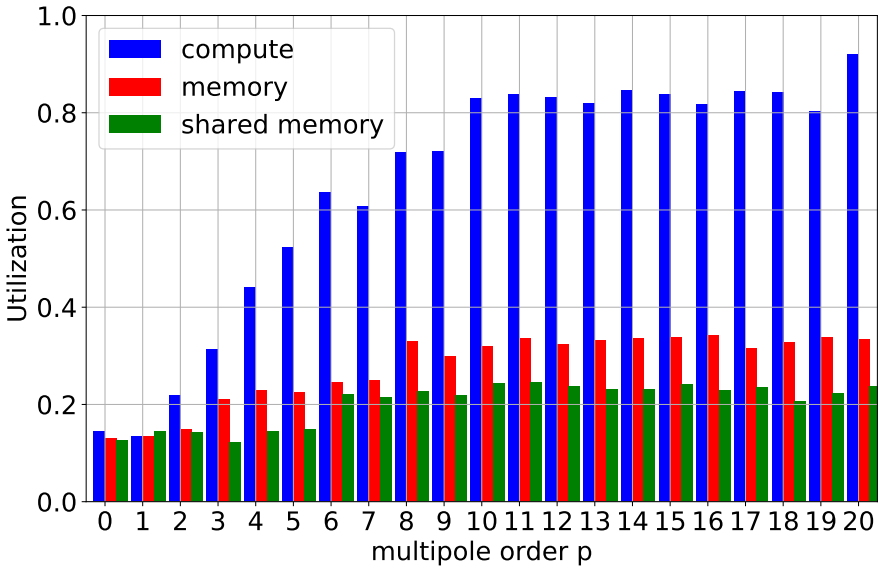


Figure 14: Hardware utilization of the symmetrical M2L kernel of the GPU-FMM.

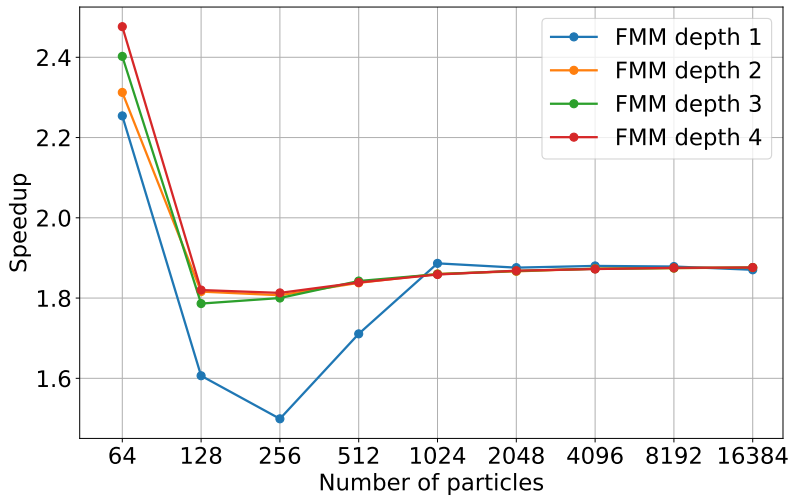


Figure 15: Speedup of calculating the P2P direct interactions in chunks of $M \times N = 32$ (i.e. for cluster pairs of size M and N) compared to computing them for all atomic pairs (i.e. for "clusters" of size $M = N = 1$). All needed FMM box-box interactions are taken into account.

where $\mathring{\omega}_{lm}^{i}$ is a chargeless multipole moment of particle at position x_i and N_{box} the number of particles in the box. The complexity of each operation is $\mathcal{O}(p^2)$. This stage is similar to P2M since the chargeless moments need to be evaluated for each particle using the same routine for a charge of $q = 1$. The performance is limited by register requirement but like in the P2M stage it runs concurrently for each particle and it is overlapped with the asynchronous memory transfer from device to host.

P2P – Particle To Particle

The FMM computes direct Coulomb interactions only for particles in the leaves of the octree and between particles in boxes that are direct neighbors. These interactions can be computed for each pair of atoms directly by starting one thread for each target particle in the box that sequentially loops over all source particles. An alternative way that better fits the GPU hardware is to compute these interactions for pairs of clusters of size M and N particles, with $M \times N = 32$ the CUDA warp size, as laid out in [41]. The forces acting on the sources and on the targets are calculated simultaneously. The interactions are computed in parallel between all needed box-box pairs in the octree. The resulting speedup of computing all atomic interactions between pairs of clusters instead of using simpler, but longer loops over pairs of atoms is shown in Fig. 15. The P2P kernels are clearly compute bound. The exact performance evaluation of the kernel can be found in [41].

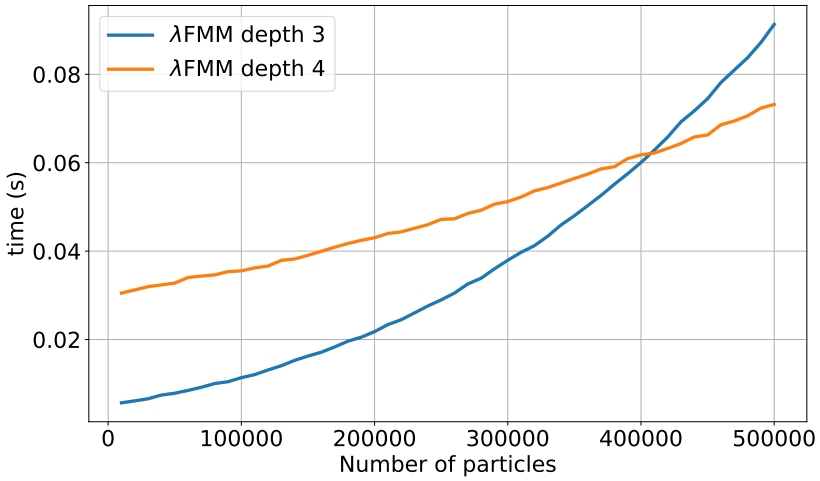


Figure 16: Absolute runtime of the λ-FMM CUDA implementation. For this example we use one λ site per 4,000 particles as estimated from the hen egg white lysozyme model system for constant-pH simulation. Each form of a λ site contains ten particles. The tests were run on a GTX 1080Ti GPU.

3.5 GPU FMM with λ-dynamics support

In addition to the regular Coulomb interactions, with λ-dynamics, extra energy terms for all forms of all λ sites need to be evaluated such that the forces on the λ particles can be derived. The resulting additional operations exhibit a very unstructured pattern that varies depending on the distribution of the particles associated with λ sites. Such a pattern can be described by multiple sparse FMM octrees that additionally interact with each other. The sparsity that emerges from a relatively small size of the λ sites necessitates a different parallelization than for a standard FMM. To support λ-dynamics efficiently, all stages of the algorithm were adapted. Especially, the most compute intense shifting (M2M, L2L) and transformation (M2L) operations need a different parallelization than that of the normal FMM to run efficiently for a sparse octree. Fig. 16 shows the runtime of the CUDA parallelized λ-FMM as a function of the system size, whereas Fig. 17 shows the overhead associated with λ-dynamics. The overhead that emerges from addition of λ sites to the simulation system scales linearly with the number of additional sites with a factor of about 10^{-3} per site. This shows that the FMM tree structure fits particularly well the λ-dynamics requirements for flexibility to compute the highly unstructured, additional particle-particle interactions. Note that our λ-FMM kernels still have the potential for more optimizations (at the moment they achieve only about 60% of the efficiency of the regular FMM kernels) such that for the final optimized implementation we expect the costs for the additional sites to be even smaller than what is shown in Fig. 17.

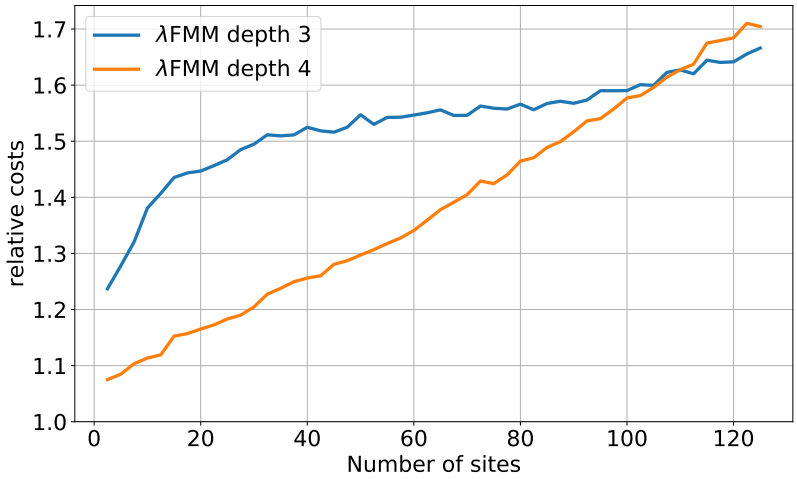


Figure 17: As Fig. 16, but now showing relative costs of adding λ-dynamics functionality to the regular GPU FMM.

4 Conclusions and Outlook

All-atom, explicit solvent biomolecular simulations with λ-dynamics are still limited to comparatively small simulation systems (<100,000 particles) and/or short timescales [18, 7, 9]. To ultimately allow for a realistic (e.g., const-pH) treatment of large biomolecular systems on long timescales, we are developing an efficient FMM that computes the long-range Coulomb interactions, including local charge alternatives for a large number of sites, with just a small overhead compared to the case without λ-dynamics.

Our FMM library is a modern C++11 based implementation tailored towards the specific requirements of biomolecular simulation, which are a comparatively small number of particles per compute core and a very short wall clock time per iteration. The presented implementation offers near-optimal performance on various SIMD architectures, an efficient CUDA version for GPUs, and it makes use of fractional tree depths for optimal performance. In addition to supporting chemical variability via λ-dynamics, it has several more unique features such as a rigorous error control, and based upon that, an automatic performance optimization at runtime. The energy drift resulting from errors in the FMM calculation can be reduced to virtually zero with a newly developed scheme that adapts the multipole expansion order p locally and on the fly in response to the requested maximum energy error. With fixed p, using multipole orders $10 - 14$ yields drifts that are smaller than those observed for typical simulations with PME. We expect the FMM to be useful also for normal MD simulations, as a drop-in PME replacement for extreme scaling scenarios where PME reaches its scaling limit.

The GPU version of our FMM will implicitly use the same parallelization framework as the CPU version. In fact, GPUs will be treated as one of several resources a node offers (in addition to CPUs), to which tasks can be scheduled. As our GPU

implementation is not a monolithic module, it can be used to calculate individual parts of the FMM, like the near-field contribution or the M2L operations of one of the local boxes only, in a fine-grained manner. How much work is offloaded to local GPUs will depend on the node specifications and on how much GPU and CPU processing power is available.

The λ-dynamics module allows to choose between three different variants of λ-dynamics. The dynamics and equilibrium distributions of the lambdas can be flexibly tuned by a barrier potential, whereas buffer sites ensure system neutrality in periodic boundary conditions. Compared to a regular FMM calculation without local charge alternatives, the GPU-FMM with λ-dynamics is only a factor of two slower even for a large (500,000 atom) simulation system with more than hundred protonatable sites.

Although some infrastructure that is needed for out-of-the-box constant-pH simulations in GROMACS still has to be implemented, with the λ-dynamics and FMM modules, the most important building blocks are in place and performing well. The next steps will be to carry out realistic tests with the new λ-dynamics implementation and to thoroughly compare to known results from older studies, before advancing to larger, more complex simulation systems that have become feasible now.

Acknowledgements

This work is supported by the German Research Foundation (DFG) Cluster of excellence *Multiscale Imaging* and under the DFG priority programme 1648 *Software for Exascale Computing (SPPEXA)*.

References

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

[2] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, and G. Sutmann. Comparison of scalable fast methods for long-range interactions. *Phys. Rev. E*, 88(6):063308, 2013.

[3] H. Berendsen, J. Grigera, and T. Straatsma. The missing term in effective pair potentials. *J. Phys. Chem.*, 91(24):6269–6271, 1987.

[4] L. V. Bock, C. Blau, A. C. Vaiana, and H. Grubmüller. Dynamic contact network between ribosomal subunits enables rapid large-scale rotation during spontaneous translocation. *Nucleic Acids Res.*, 43(14):6747–6760, 2015.

[5] M. Bolten, F. Fahrenberger, R. Halver, F. Heber, M. Hofmann, I. Kabadshow, O. Lenz, M. Pippig, and G. Sutmann. ScaFaCoS, C subroutine library. URL `http://scafacos.github.com`.

[6] H. Dachsel. An error-controlled fast multipole method. *J. Chem. Phys.*, 132: 119901, 2010. doi: 10.1063/1.3264952.

[7] P. Dobrev, S. Donnini, G. Groenhof, and H. Grubmüller. Accurate three states model for amino acids with two chemically coupled titrating sites in explicit solvent atomistic constant pH simulations and pKa calculations. *Journal of Chemical Theory and Computation*, 13(1):147–160, 2017. doi: 10.1021/acs.jctc.6b00807.

[8] S. Donnini, F. Tegeler, G. Groenhof, and H. Grubmüller. Constant pH molecular dynamics in explicit solvent with λ-dynamics. *J. Chem. Theory Comput.*, 7:1962–1978, 2011. doi: 10.1021/ct200061r.

[9] S. Donnini, R. T. Ullmann, G. Groenhof, and H. Grubmüller. Charge-neutral constant pH molecular dynamics simulations using a parsimonious proton buffer. *J. Chem. Theory Comput.*, 12(3):1040–1051, 2016. doi: 10.1021/acs.jctc.5b01160.

[10] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick. A communication-optimal n-body algorithm for direct interactions. *Parallel and Distributed Processing Symposium, International*, 0:1075–1084, 2013. ISSN 1530-2075. doi: 10.1109/IPDPS.2013.108.

[11] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen. A smooth particle mesh Ewald method. *J. Chem. Phys.*, 103(19):8577–8593, 1995. doi: 10.1063/1.470117.

[12] A. G. Garcia, A. Beckmann, and I. Kabadshow. *Accelerating an FMM-Based Coulomb Solver with GPUs*, pages 485–504. Springer International Publishing, Cham, 2016. ISBN 978-3-319-40528-5. doi: 10.1007/978-3-319-40528-5_22.

[13] G. B. Goh, J. L. Knight, and C. L. Brooks. Constant pH molecular dynamics simulations of nucleic acids in explicit solvent. *J. Chem. Theory Comput.*, 8: 36–46, 2012. doi: 10.1021/ct2006314.

[14] G. B. Goh, B. S. Hulbert, H. Zhou, and C. L. Brooks III. Constant pH molecular dynamics of proteins in explicit solvent with proton tautomerism. *Proteins: structure, function, and bioinformatics*, 82(7):1319–1331, 2014.

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

[16] Z. Guo, C. Brooks, and X. Kong. Efficient and flexible algorithm for free energy calculations using the λ-dynamics approach. *J. Phys. Chem. B*, 102(11):2032–2036, 1998.

[17] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.*, 4:435–447, 2008. doi: 10.1021/ct700301q.

[18] Y. Huang, W. Chen, J. A. Wallace, and J. Shen. All-atom continuous constant pH molecular dynamics with particle mesh Ewald and titratable water. *J. Chem. Theory Comput.*, 12(11):5411–5421, 2016.

[19] J. S. Hub, B. L. de Groot, H. Grubmüller, and G. Groenhof. Quantifying artifacts in Ewald simulations of inhomogeneous systems with a net charge. *J. Chem. Theory Comput.*, 10:381–390, 2014. doi: 10.1021/ct400626b.

[20] M. Igaev and H. Grubmüller. Microtubule assembly governed by tubulin allosteric gain in flexibility and lattice induced fit. *eLife*, 7:e34353, 2018.

[21] J. Jung, W. Nishima, M. Daniels, G. Bascom, C. Kobayashi, A. Adedoyin, M. Wall, A. Lappala, D. Phillips, W. Fischer, C.-S. Tung, T. Schlick, Y. Sugita, and K. Y. Sanbonmatsu. Scaling molecular dynamics beyond 100,000 processor cores for large-scale biophysical simulations. *J. Comput. Chem.*, 2019.

[22] I. Kabadshow and H. Dachsel. The Error-Controlled Fast Multipole Method for Open and Periodic Boundary Conditions. In G. Sutmann, P. Gibbon, and T. Lippert, editors, *Fast Methods for Long-Range Interactions in Complex Systems*, volume 6 of *IAS Series*, pages 85–114. FZ Jülich, 2011.

[23] J. Khandogin and C. L. Brooks. Constant pH molecular dynamics with proton tautomerism. *Biophys. J*, 89(1):141–157, 2005.

[24] J. G. Kirkwood. Statistical mechanics of fluid mixtures. *J. Chem. Phys.*, 3(5): 300–313, 1935.

[25] J. L. Knight and C. L. Brooks III. λ-dynamics free energy simulation methods. *J. Comput. Chem.*, 30(11):1692–1700, 2009.

[26] J. L. Knight and C. L. Brooks III. Applying efficient implicit nongeometric constraints in alchemical free energy simulations. *J. Comput. Chem.*, 32(16): 3423–3432, 2011. doi: 10.1002/jcc.21921.

[27] X. Kong and L. Brooks III, Charles. λ-dynamics: A new approach to free energy calculations. *J. Chem. Phys.*, 105:2414–2423, 1996. doi: 10.1063/1.472109.

[28] W. Kopec, D. A. Köpfer, O. N. Vickery, A. S. Bondarenko, T. L. Jansen, B. L. de Groot, and U. Zachariae. Direct knock-on of desolvated ions governs strict ion selectivity in K+ channels. *Nat. Chem.*, 10(8):813, 2018.

[29] C. Kutzner, D. van der Spoel, M. Fechner, E. Lindahl, U. W. Schmitt, B. L. de Groot, and H. Grubmüller. Speeding up parallel GROMACS on high-latency networks. *J. Comput. Chem.*, 28(12):2075–2084, 2007. doi: 10.1002/jcc.20703.

[30] C. Kutzner, R. Apostolov, B. Hess, and H. Grubmüller. Scaling of the GROMACS 4.6 molecular dynamics code on SuperMUC. In M. Bader, A. Bode, and H.-J. Bungartz, editors, *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, pages 722–730. IOS Press, Amsterdam/Netherlands, 2014. doi: 10.3233/978-1-61499-381-0-722.

[31] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. de Groot, and H. Grubmüller. Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. *J. Comput. Chem.*, 36(26):1990–2008, 2015. doi: 10.1002/jcc.24030.

[32] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller. More bang for your buck: Improved use of GPU nodes for GROMACS 2018. *J. Comput. Chem.*, 40(27):2418–2431, 2019. doi: 10.1002/jcc.26011.

[33] M. S. Lee, F. R. Salsbury Jr, and C. L. Brooks III. Constant-pH molecular dynamics using continuous titration coordinates. *Proteins Struct. Funct. Bioinf.*, 56(4):738–752, 2004.

[34] E. Lindahl, M. Abraham, B. Hess, and D. van der Spoel. GROMACS 2019.3 Manual. *Zenodo*, Jun 2019. doi: 10.5281/zenodo.3243834.

[35] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 9(1):21–65, 1991.

[36] D. J. Mermelstein, C. Lin, G. Nelson, R. Kretsch, J. A. McCammon, and R. C. Walker. Fast and flexible GPU accelerated binding free energy calculations within the AMBER molecular dynamics package. *J. Comput. Chem.*, 39(19): 1354–1358, 2018.

[37] J. E. Mertz and B. M. Pettitt. Molecular dynamics at a constant pH. *The International Journal of Supercomputer Applications and High Performance Computing*, 8(1):47–53, 1994.

[38] D. L. Mobley and P. V. Klimovich. Perspective: Alchemical free energy calculations for drug discovery. *J. Chem. Phys.*, 137(23):230901, 2012.

[39] J. Mongan and D. A. Case. Biomolecular simulations at constant pH. *Curr. Opin. Struct. Biol.*, 15(2):157–163, 2005.

[40] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2019. Version 10.1.243.

[41] S. Páll and B. Hess. A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput. Phys. Commun.*, 184:2641–2650, 2013. doi: 10.1016/j.cpc.2013.06.003.

[42] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl. Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In S. Markidis and E. Laure, editors, *Solving Software Challenges for Exascale*, pages 3–27, Cham, 2015. Springer International Publishing. ISBN 978-3-319-15976-8.

[43] J. R. Perilla, B. C. Goh, C. K. Cassidy, B. Liu, R. C. Bernardi, T. Rudack, H. Yu, Z. Wu, and K. Schulten. Molecular dynamics simulations of large macromolecular complexes. *Curr. Opin. Struct. Biol.*, 31:64–74, 2015.

[44] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007. ISBN 0521880688, 9780521880688.

[45] R. Schulz, B. Lindner, L. Petridis, and J. C. Smith. Scaling of multimillion-atom biological molecular dynamics simulation on a petascale supercomputer. *J. Chem. Theory Comput.*, 5(10):2798–2808, 2009.

[46] D. Seeliger and B. L. De Groot. Protein thermostability calculations using alchemical free energy simulations. *Biophys. J.*, 98(10):2309–2316, 2010.

[47] M. R. Shirts, D. L. Mobley, and J. D. Chodera. Alchemical free energy calculations: ready for prime time? *Annual reports in computational chemistry*, 3: 41–59, 2007.

[48] J. A. Wallace and J. K. Shen. Charge-leveling and proper treatment of long-range electrostatics in all-atom molecular dynamics at constant pH. *J. Chem. Phys.*, 137(18):184105, 2012.

[49] C. A. White and M. Head-Gordon. Fractional tiers in fast multipole method calculations. *Chem. Phys. Lett.*, 257(5-6):647–650, 1996. ISSN 0009-2614. doi: 10.1016/0009-2614(96)00574-X.

[50] R. Yokota and L. A. Barba. A Tuned and Scalable Fast Multipole Method as a Preeminent Algorithm for Exascale Systems. *CoRR*, abs/1106.2176, 2011. URL `http://arxiv.org/abs/1106.2176`.

II.2 A CUDA Fast Multipole Method with highly efficient M2L far field evaluation

In this paper, I reported on the GPU parallelization process of the most performance critical operator of the FMM. The entire work presented in this paper was conducted by me. I also completely wrote the paper. The content of this work was then finally edited by Dr. Carsten Kutzner, Prof. Dr. Gert Lube and Prof. Dr. Helmut Grubmüller. Dr. Carsten Kutzner also enhanced graphically a few figures in this work. This paper has been accepted for publication in The International Journal of High Performance Computing Applications (IJHPCA).

A CUDA Fast Multipole Method with highly efficient M2L far field evaluation

Bartosz Kohnke[1], Carsten Kutzner[1], Andreas Beckmann[2], Gert Lube[3], Ivo Kabadshow[2], Holger Dachsel[2], and Helmut Grubmüller[1]

[1]*Max Planck Institute for Biophysical Chemistry, Theoretical and Computational Biophysics, Am Faßberg 11, 37077 Göttingen*
[2]*Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm Johnen Straße 1, 52428 Jülich*
[3]*Institute for Numerical and Applied Mathematics, Georg-August University of Göttingen, Lotzestr. 16–18, 37083 Göttingen*

Solving an N-body problem, electrostatic or gravitational, is a crucial task and the main computational bottleneck in many scientific applications. Its direct solution is an ubiquitous showcase example for the compute power of graphics processing units (GPUs). However, the naïve pairwise summation has $\mathcal{O}(N^2)$ computational complexity. The fast multipole method (FMM) can reduce runtime and complexity to $\mathcal{O}(N)$ for any specified precision. Here, we present a CUDA-accelerated, C++ FMM implementation for multi particle systems with r^{-1} potential that are found, e.g., in biomolecular simulations. The algorithm involves several operators to exchange information in an octree data structure. We focus on the Multipole-to-Local (M2L) operator, as its runtime is limiting for the overall performance. We propose, implement and benchmark three different M2L parallelization approaches. Approach (1) utilizes Unified Memory to minimize programming and porting efforts. It achieves decent speedups for only little implementation work. Approach (2) employs CUDA Dynamic Parallelism to significantly improve performance for high approximation accuracies. The presorted list-based approach (3) fits periodic boundary conditions particularly well. It exploits FMM operator symmetries to minimize both memory access and the number of complex multiplications. The result is a compute bound implementation, i.e., performance is limited by arithmetic operations rather than by memory accesses. The complete CUDA parallelized FMM is incorporated within the GROMACS molecular dynamics package as an alternative Coulomb solver.

1 Introduction

The fast multipole method (FMM) was introduced by Greengard and Rokhlin [23] to efficiently evaluate pairwise, Coulombic or gravitational, interactions in many body systems, which arise in many diverse fields like biomolecular simulation [16, 27], astronomy [47, 6] or plasma physics [13]. Moreover, the FMM can improve iterative solvers for integral equations by speeding up the underlying matrix-vector products [18, 25].

The originally proposed FMM uses a spherical harmonics representation of the inverse distance r^{-1} between particles. For distant interactions (far field), which can be strictly defined, it uses multipole expansions built by clustered particle groups. The expansions are shifted and then transformed into Taylor moments by applying linear operators in a hierarchical manner to achieve linear scaling with respect to the number of particles. The complexity of the operators is $\mathcal{O}(p^4)$, where p is the order of the multipole expansion. White and Head-Gordon [56] and Greengard and Rokhlin [24] proposed rotational operators, that align the transformation axis to reduce the operator complexity to $\mathcal{O}(p^3)$. Cheng et al. [11] used plain wave expansions to further reduce the complexity of the operators to $\mathcal{O}(p^2)$, however a few $\mathcal{O}(p^3)$ translations are still required. The algorithm has been developed further to support oscillatory kernels e^{ikr}/r [58]. Fong and Darve [20] parametrized the inverse distance function using Chebyshev polynomials and proposed a "black-box" FMM, which uses a minimal number of coefficients to represent the far field.

In atomistic molecular dynamics (MD) simulations, Newton's equations of motion are solved for a system of N particles [3] in a potential that accounts for all relevant interactions between the atoms. The integration time step is limited to a few femtoseconds such that the fastest atomic motions can be resolved. To reach the time scales many biomolecules operate on, millions of time steps need to be computed [38, 10, 50, 46]. This can easily require weeks or even months of compute time even on modern hardware [37]. Hence, to speed up the calculation of an MD trajectory, the execution time for each individual time step has to be reduced. This can be achieved with better algorithms, with special-purpose hardware [52], by introducing heterogeneous parallelization, e.g. harnessing SIMD, multi-core, and multi-node parallelism [28, 1, 44] and by using GPUs [48, 43]. Here, we utilize GPUs for our FMM implementation.

The electrostatic contribution to the inter-atomic forces is governed by Coulomb's law

$$\mathbf{F}_{ij} = \frac{1}{4\pi\epsilon_0} \frac{q_i q_j}{\|\mathbf{r}_{ij}\|_2^2} \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|_2}, \tag{1}$$

where $\mathbf{r}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is a vector distance between atoms carrying partial charges q_i, q_j at positions $\mathbf{x}_i$, $\mathbf{x}_j$, ϵ_0 is the vacuum permittivity and $\|\cdot\|_2$ is the Euclidean distance. The calculation of nonbonded forces, i.e. Coulomb and van der Waals forces, is usually by far the most time-consuming part of an MD step. The van der

Waals forces decay very quickly with distance r, so calculating them up to a cutoff distance suffices. The Coulomb forces, however, decay only quadratically with r, and the use of a finite Coulomb cutoff can therefore lead to severe simulation artifacts [49, 45]. Direct evaluation of the electrostatic interactions in a typical biomolecular simulation system becomes prohibitive for two reasons. First, the $\mathcal{O}(N^2)$ scaling of a direct evaluation hinders its usage already at small system sizes, e.g. for $N \approx 50,000$ particles. Second, the usually employed periodic boundary conditions (PBC) make such calculation even impossible. Biomolecular simulation, therefore, requires an efficient Coulomb solver that properly accounts for the full, long range nature of the electrostatic interactions.

To this aim, several FMM implementations have been developed. A standard FMM was included as an electrostatic solver for the NAMD package [8, 40]. Ding et al. [14] proposed the Cell Multipole Method (CMM) to simulate polymer systems of up to 1.2 million atoms. In further work, they combined CMM with the Ewald method showing a considerable speedup with respect to a pure Ewald treatment [15]. Niedermeier and Tavan [42] introduced a structure-adapted multipole method. Eichinger et al. [17] combined the structure-adapted multipole method with a multiple-time-step algorithm. Andoh et al. [4] developed MODYLAS, a FMM adoption for very large MD systems and benchmarked it on the K-computer using 65,536 nodes. Very recently, it was extended to support rectangular boxes [5]. Yoshii et al. [60] developed a FMM for MD systems with two-dimensional periodic boundary conditions. Shamshirgar et al. [51] implemented a regularization method for improved FMM energy conservation. Gnedin [22] combined fast Fourier transforms (FFTs) and the FMM for improved performance.

Considering efficient parallelization approaches, Gumerov and Duraiswami [26] pioneered the GPU implementations of the spherical harmonics FMM with rotational operators. Depending on accuracy they achieved speedups of 30–70 with respect to a single CPU. Different GPU parallelization schemes for the "black-box" FMM [20] were implemented by Takahashi et al. [53]. Yokota et al. [59] parallelized ExaFmm on a GPU cluster with 32 GPUs achieving a parallel efficiency of 44 % and 66 % for 10^6 and 10^7 particles, respectively. Even more GPUs (256) were used by Lashuk et al. [39] for a system of 256 million particles. Rotational based Multipole-to-Local operators were efficiently parallelized with GPUs by Garcia et al. [21]. Task based parallelization approaches to the FMM were proposed in Blanchard et al. [7] and Agullo et al. [2]. A review of fast multipole techniques for calculation of electrostatic interactions in MD systems can be found in Kurzak and Pettitt [34].

However, the early adoptions of FMMs in MD simulation codes were mostly superseded by particle Mesh Ewald (PME) [19] due to its higher single-node performance. As a result, PME currently dominates the field. It is based on the FFT, which inherently provides the PBC solution. Nevertheless, PME suffers from a scaling bottleneck when parallelized over many nodes, as the underlying FFTs require all-to-all communication [9, 35, 36]. In addition, large systems with nonuniform particle distributions become memory intensive, since PME evaluates the forces on a uniform mesh across the whole computational domain.

In the era of ever-increasing parallelism and exascale computers, it is time to revisit the FMM, which does not suffer from the above mentioned limitations. To this end, we implemented and benchmarked a single-node full CUDA parallel FMM. Our implementation has been tailored for MD simulations, i.e. it targets a millisecond order runtime for one MD step by careful GPU parallelization of all FMM stages and by optimizing their flow to hide possible latencies. It was also meticulously integrated into the GROMACS package to avoid additional FMM independent performance bottlenecks. Here, we present three different parallelization approaches. The implementation is based on the ScaFaCos FMM [6], which utilizes spherical harmonics to describe the r^{-1} function. We use octree grouping to describe the interaction hierarchy. Such grouping is achieved by recursive subdivision of the cubic simulation box into eight equal subboxes. It has a major advantage: the far field operators can be precomputed for the whole simulation box, allowing for efficient parallelization. Additionally, the PBC computation becomes negligible as the PBC operators reduce to a single operator appliance. Moreover, a strict error control of the approximation [12] can be applied.

Here, we focus on the CUDA parallelization of the Multipole-to-Local (M2L) operator, which is most limiting to the overall FMM far field performance. An overview of the parallelized FMM, including all stages and complete runtimes, can be found in Kohnke et al. [33].

2 The Fast Multipole Method

We consider a system of $N \gg 1$ particles. Following Hockney and Eastwood [29], the challenge is to most efficiently evaluate

$$\Phi(\mathbf{x}_j) = \sum_{\substack{k=0 \\ k \neq j}}^{N-1} \frac{q_k}{\|\mathbf{x}_j - \mathbf{x}_k\|_2}, \; j = 0, ..., N-1, \tag{2}$$

where $\mathbf{x}_j$ and $\mathbf{x}_k$ are positions of particles j and k, respectively and q_k is the charge of the k-th particle. For a direct solution of Eq. (2), interactions between all pairs of particles (j, k) with $j \neq k$ need to be computed. This leads to two nested loops and $\mathcal{O}(N^2)$ calculation steps.

2.1 Mathematical foundations

Expansion of the inverse distance between arbitrary particles $\mathbf{x}_j$ and $\mathbf{x}_k$, $j \neq k$ yields

$$\frac{1}{\|\mathbf{x}_j - \mathbf{x}_k\|_2} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{\|\mathbf{x}_j\|_2^l}{\|\mathbf{x}_k\|_2^{l+1}} Y_{lm}^*(\theta_j, \phi_j) Y_{lm}(\theta_k, \phi_k), \tag{3}$$

where

$$Y_{lm}(\theta, \phi) := \sqrt{\frac{2l+1}{4\pi}} \sqrt{\frac{(l-m)!}{(l+m)!}} P_{lm}(\cos\theta) e^{im\phi} \tag{4}$$

are spherical harmonics and Y^* their complex-conjugate, θ and ϕ are polar and azimuthal angle, respectively, and P_{lm} are the associated Legendre polynomials

$$P_{lm}(y) := (-1)^m (1-y^2)^{m/2} \frac{d^m}{dy^m} P_l(y), \tag{5}$$

where P_l are ordinary Legendre polynomials

$$P_l(y) := \frac{1}{2^l l!} \frac{d^l}{dy^l} \left(y^2 - 1\right)^l. \tag{6}$$

The normalized associated Legendre polynomials form an orthonormal set of basis functions on the surface of a sphere. The j-th and k-th dependent parts of the right hand side of Eq. (3) are chargeless multipole moments

$$\mathring{\omega}_{lm}^j := \mathring{\omega}_{lm}^j(\mathbf{x}_j) := \frac{\|\mathbf{x}_j\|_2^l}{(l+m)!} P_{lm}(\cos\theta_j) e^{im\phi_j}, \tag{7}$$

and chargeless local moments

$$\mathring{\mu}_{lm}^k := \mathring{\mu}_{lm}^k(\mathbf{x}_k) := \frac{(l-m)!}{\|\mathbf{x}_k\|_2^{l+1}} P_{lm}(\cos\theta_k) e^{im\phi_k}, \tag{8}$$

respectively. The moments weighted with corresponding charges q_j and q_k, respectively, can be summed yielding charged multipole moments

$$\omega_{lm} := \sum_{j=0}^{J-1} q_j \mathring{\omega}_{lm}^j, \tag{9}$$

and charged local moments

$$\mu_{lm} := \sum_{k=0}^{K-1} q_k \mathring{\mu}_{lm}^k. \tag{10}$$

This allows to evaluate the potential at arbitrary particles at $\mathbf{x}_j$, $j = 0, ..., J-1$ due to a distant discrete charge distribution of K particles with positions $\mathbf{x}_k$, $k = 0, ..., K-1$ in terms of charged local moments and chargeless multipole moments with

$$\Phi(\mathbf{x}_j) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \mu_{lm} \mathring{\omega}_{lm}^{j}. \tag{11}$$

This calculation is referred to as far field. To achieve convergence in Eq. (3),

$$\|\mathbf{x}_j\|_2 < \|\mathbf{x}_k\|_2 \tag{12}$$

has to be fulfilled for all distinct index pairs j and k. Application of addition theorems for regular and irregular solid harmonics [54] yields translation and transformation operators for the expansions. The moments ω_{lm} of a multipole expansion about a common origin $\mathbf{a}$

$$\omega(\mathbf{a}) := \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \omega_{lm} \tag{13}$$

of particles $\mathbf{x}_j$, $j = 0, ..., J-1$ can be translated to a new origin $\mathbf{a}'$ with

$$\omega_{lm}(\mathbf{a}') = \sum_{j=0}^{l} \sum_{k=-j}^{j} \omega_{jk}(\mathbf{a}) \mathbf{A}_{l-j,m-k}(\mathbf{a} - \mathbf{a}'), \tag{14}$$

where $\mathbf{A} \equiv \mathring{\omega}$ is the Multipole-to-Multipole operator. Further, the multipole expansion $\omega(\mathbf{a})$ can be transformed into a local expansion $\mu(\mathbf{r})$ at $\mathbf{r}$ with $\|\mathbf{r}\|_2 > \|\mathbf{x}_j\|_2$, $j = 0, ..., J-1$ with

$$\mu_{lm}(\mathbf{r}) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \omega_{jk}(\mathbf{a}) \mathbf{M}_{l+j,m+k}(\mathbf{a} - \mathbf{r}), \tag{15}$$

where $\mathbf{M} \equiv \mathring{\mu}$ is the Multipole-to-Local operator. Finally, the local expansions $\mu(\mathbf{r})$ can be translated to any point $\mathbf{r}'$ with

$$\mu_{lm}(\mathbf{r}') = \sum_{j=l}^{\infty} \sum_{k=-j}^{j} \mu_{jk}(\mathbf{r}) \mathbf{C}_{j-l,k-m}(\mathbf{r} - \mathbf{r}'), \tag{16}$$

where $\mathbf{C} \equiv \mathring{\omega}$ is the Local-to-Local operator.

2.2 Algorithm

Applying the operators defined in the previous section requires truncation of Eq. (3) to a finite multipole order p, which controls the accuracy of the solution approximation. Such expansions have a triangular shape with indexing shown in

Fig. 1. The truncation yields

$$\omega, \mu, \mathbf{A}, \mathbf{C} \in \mathbb{K}^{p\times p} := \{(a_{lm})_{l=0,\dots,p,\ m=-l,\dots,l} \mid a_{lm} \ \in \mathbb{C}\} \tag{17}$$

and $\mathbf{M} \in \mathbb{K}^{2p\times 2p}$.

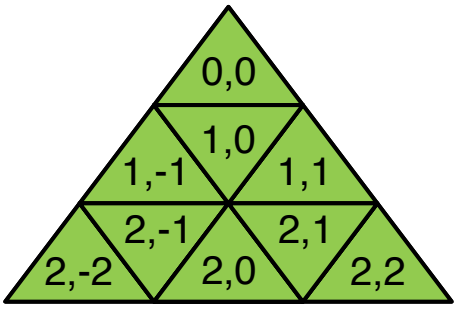


Figure 1: Indexing of triangular shaped matrices. The used indexing scheme is based on standard matrix index notation: The first subscript is a row number, the second one is a column number, which can be negative. In case of (l, m) notation, $l \geq |m|$ and $l \leq p$.

The translations and transformations defined in section 2.1 are performed on moments expanded for clusters of particles. The clustering is based on a hierarchical partition of the computational domain $\Omega := [0, \ell]^3 \in \mathbb{R}^3$ into 8^d boxes for $d = 0, ..., \mathcal{D}$, where $\mathcal{D}$ is a predefined parameter. It leads to an octree of depth $\mathcal{D}$ shown in Fig. 2. Fig. 3 illustrates the six main stages of the FMM and their execution order. In

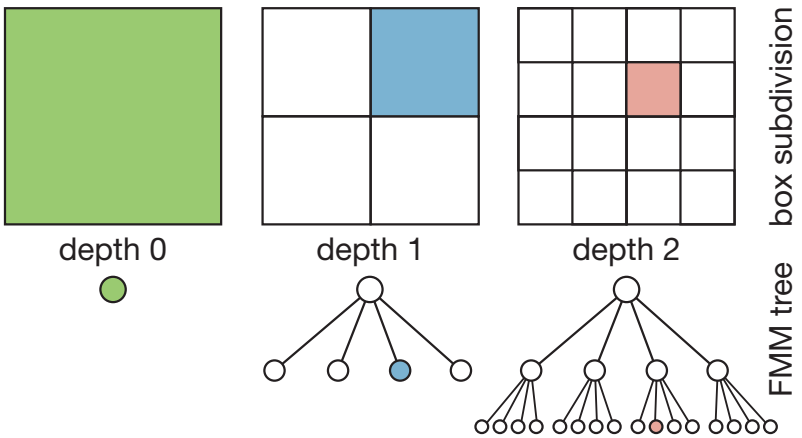


Figure 2: 2D example of FMM tree depth and resulting box subdivision for depths $\mathcal{D} = 0$, 1, and 2. At $\mathcal{D} = 1$, the whole simulation box (green) is split into four sub-boxes (blue). At $\mathcal{D} = 2$, each of the sub-boxes is split again into four smaller boxes (red).

the Particle-to-Multipole (P2M) stage, the particles occupying boxes on the deepest

level $\mathcal{D}$ of the octree are expanded to multipoles ω with respect to the center of the boxes. In the Multipole-to-Multipole (M2M) stage, the expanded moments ω are distributed to all boxes of the octree by translating them level-wise from d to $d-1$, $d = \mathcal{D}, ..., 1$ with the $\mathbf{A}$ operator. Subsequently, during the Multipole-to-Local (M2L) stage, the multipole moments ω are transformed into local moments μ by applying the operator $\mathbf{M}$. A detailed description follows in the next section. After M2L, in the Local-to-Local (L2L) stage, the local moments μ are shifted from the octree root to the leaves with the $\mathbf{C}$ operator. The interactions between particles occupying the same lowest level boxes and between neighboring boxes (near field) are evaluated directly (P2P), Eq. (2). Finally, the far field forces and potentials are evaluated at particle positions in the tree leaves. Additionally, the number of directly interacting boxes can be defined with a `well separation criterion`, which controls how many layers of adjacent boxes interact directly on the lowest tree depth. In the following we will only discuss the case with one well separated layer of boxes.

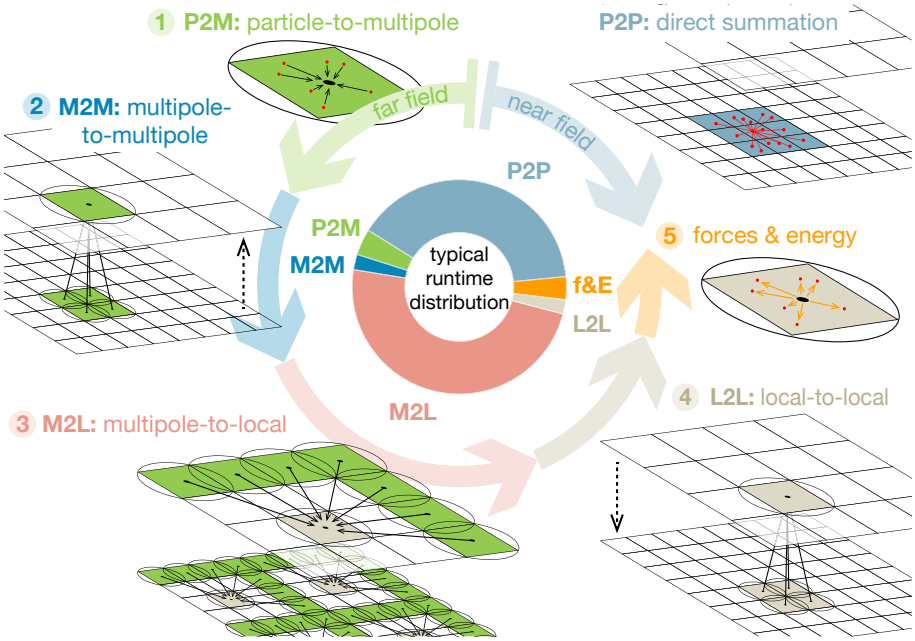


Figure 3: The six different stages of the FMM with an exemplary execution time distribution at the center. The near field part (P2P, top right corner) can be executed concurrently with the far field (stages 1–5) in a parallel implementation. Green squares indicate the representation by multipoles, light brown squares a representation by local moments, blue squares indicate direct summation.

2.3 The Multipole-to-Local (M2L) transformation

We will now explain the the M2L transformation and its execution hierarchy. Let $\mathcal{D}$ be a fixed depth of an octree and p a multipole order. The multipole or local expansions in box i at depth $d = 0, ..., \mathcal{D}$ will be denoted by ω_i^d and μ_i^d, respectively. For each μ_i^d there exists an interaction set $\mathcal{L}_i^d$, $|\mathcal{L}_i^d| = 208$. Fig. 4 shows the interaction set $\mathcal{L}_i^d$, which contains the indices of multipole expansions ω_j^d in all children boxes of the direct neighbors of μ_i^d's parent box. The direct neighbors share at least one common vertex, edge or face with each other. A particular μ_i^d is calculated from all ω_j^d, $j \in \mathcal{L}_i^d$, omitting μ_i^d's direct neighbor boxes in order to satisfy Eq. (12). This results in 189 $\mathcal{O}(p^4)$ M2L transformations.

Let $\mathcal{M} := \left\{\mathcal{M}^d\right\}_{d=1}^{\mathcal{D}}$ be the set of level-wise operator sets $\mathcal{M}^d := \{\mathbf{M}_{j\to i}^d \,|\, \mathbf{M}$ transforms j-th multipole moment to i-th local moment at level $d\}$. All M2L operations performed in the FMM octree yield

$$\mu_i^d = \sum_{\substack{j\in\mathcal{L}_i^d \\ \mathbf{M}_j^d\in\mathcal{M}^d}} \mathbf{M}_{j\to i}^d \, \omega_j^d, \; i = 1, ..., 8^d, d = 1, ..., \mathcal{D}. \tag{18}$$

Fig. 5 shows one $\mathcal{O}(p^4)$ M2L transformation. It contains $\mathcal{O}(p^2)$ dot products between an ω and a part of the corresponding operator $\mathbf{M}$.

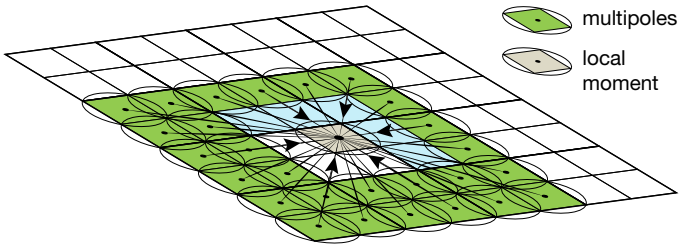


Figure 4: 2D interaction set $\mathcal{L}^d$ (green) of an arbitrary box with a local moment μ^d (light brown). The white boxes do not belong to interaction set $\mathcal{L}^d$. The interactions with the light blue boxes need to be skipped as well because they are nearest neighbors.

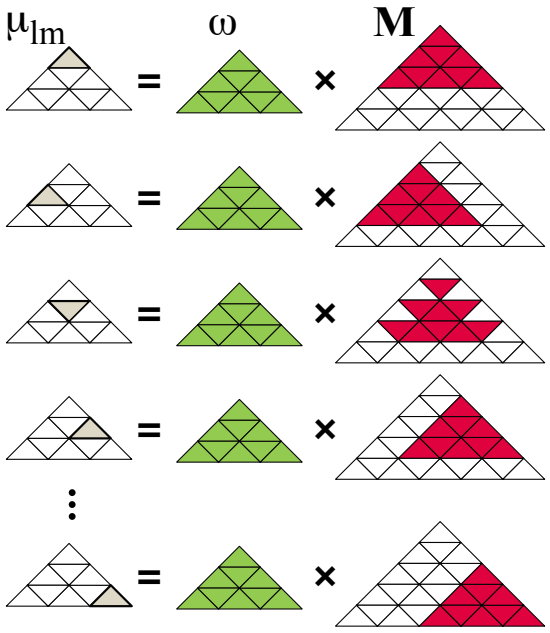


Figure 5: One M2L transformation. The matrix-vector like multiplication requires a part of the **M** operator (red) to calculate one element μ_{lm} (light brown) of a target expansion.

3 Implementation

We focus our GPU parallelization efforts on the M2L operator, as it is the most time consuming FMM far field operator (see Fig. 3 above and Fig. 12 in Kohnke et al. [33]). In PBC, it requires 189 transformations per box, whereas both M2M and L2L, which translate the moments between different tree levels, require only a single transformation per octree box (except for the root box). Since these transformations are of the same complexity, M2L involves 94.5× the number of operations as M2M and L2L combined. The second most time consuming part is the P2P near field computation, which will not be discussed in this paper, was optimized as laid out in Páll and Hess [43]. Proper choice of $\mathcal{D}$ and p allows to balance the near and far field contribution, which minimizes the overall runtime. In case of parallel implementation these stages can run concurrently.

3.1 CUDA implementation considerations

We will now briefly outline the CUDA programming model, see Nickolls et al. [41] for details. A typical GPU consists of a few thousand cores that are grouped into larger units called multiprocessors. CUDA threads are organized in `blocks`. Threads within a `block` are grouped into subunits called `warps`, each consisting of 32 threads. For optimal performance, threads within the same `warp` should execute the

same instruction, otherwise the execution is serialized. This type of parallelization is called Single Instruction Multiple Threads (SIMT). Once a `block` of threads is spawned, it occupies the multiprocessor until the respective computations are completed. Dynamic scheduling is performed `warp`-wise, thus thread `blocks` should consist of at least several `warps` to hide memory and arithmetic latencies within a multiprocessor. `Blocks` are organized in `grids`. Each `block` of a `grid` and thread of a `block` is identified with its unique 1D, 2D or 3D index. The dimensions of the `grid` and the `blocks` can be chosen independently. To identify threads, CUDA provides the 3D structures `gridDim`, `blockDim`, `blockIdx` and `threadIdx`.

We will use the following abbreviations: $B_\alpha :=$ `blockDim.`α, $G_\alpha :=$ `gridDim.`α, $Bid_\alpha :=$ `blockIdx.`α and $tid_\alpha :=$ `threadIdx.`α, $\alpha \in \{x, y, z\}$. The hierarchy of threads described above affects the memory access and communication between threads. Whereas all threads can access global memory, this access should be minimized as it has a latency of a few hundred cycles. The memory within a `block` can be shared via `shared memory`. If no bank conflicts occur fetching `shared memory` is only slightly slower than register access (20–40 cycles). Synchronization of threads is possible only within a `block`. Since CUDA-6.0, threads within the same `warp` are able to share their content via the `shuffle` instruction by directly sharing the registers.

3.2 Sequential FMM and data structures

Our CUDA implementation is based on a C++11 version of the sequential ScaFaCos FMM [6]. It provides class templates with a possibility to use diverse memory allocators. With CUDA Unified Memory [31] the usage of original data structures became feasible by harnessing the C++ memory allocators.

To allow for an efficient manipulation of triangular shaped data (see Figs. 1 and 5), we have implemented a dedicated `triangular_matrix` class that stores the moments and operators. It provides the indexing logic and utilizes a 1D vector of complex values (`std::vector<complex>`) for this purpose. For symmetry reasons, it suffices to store one half of the triangular matrix for the moments, as the entries on the left ($m < l$) and right side ($m > l$) are identical except for the signs. The signs are computed on the fly at negligible costs from the parity of indices. The overall size of the matrices depends on the multipole order p. Exploiting symmetry, $(p^2 + p)/2$ complex values are stored for the expansions and $((2p)^2 + 2p)/2$ for the $\mathbf{M}$ operator.

Let $\mathcal{D}$ be a fixed depth of the tree. Thus, there are $d = 0, ..., \mathcal{D}$ levels in an octree. For Multipole-to-Local operations, as described in *The Multipole-to-Local (M2L) transformation* Section, an underlying tree implementation is needed. Listing 1 shows a very basic approach for traversing an octree of depth $\mathcal{D}$. The function `index`(x, y, z, d) applies the lexicographic approach to compute a unique 1D box

Listing 1: Loops for traversing an octree in 3D space (pseudocode).

```
1 //chosen tree depth D
2 int d,z,y,x,i;
3 for (d = 0; d <= D; ++d)
4 {
5  for (z = 0; z < std::pow(2,d); ++z)
6  {
7   for (y = 0; y < std::pow(2,d); ++y)
8   {
9    for (x = 0; x < std::pow(2,d); ++x)
10   {
11    //unique one dimensional index
12    i = index(x,y,z,d);
```

index in the octree:

$$z\,\mathtt{dim}(d)\,\mathtt{dim}(d) + y\,\mathtt{dim}(d) + x + \mathtt{nb}(d-1), \tag{19}$$

where $\mathtt{dim}(d) := 2^d$ is the number of boxes in each orthogonal direction and $\mathtt{nb}(d) := \sum_{d=0}^{\mathcal{D}} 8^d = \lfloor (8^{d+1})/7 \rfloor$ is the number of all boxes in an octree of depth d. The parent box index is easily obtained as $\mathtt{index}(x/2,\ y/2,\ z/2,\ d-1)$.

Listing 2 shows the sequential form of the M2L transformation. The first four for-loops (lines 3 – 9) traverse the octree as shown in Listing 1. `omega` and `mu` store the pointers to `triangular_matrix` objects for the multipole and local moments, respectively. The next three for-loops determine all multipole expansions $\omega_j^d \in \mathcal{L}_i^d$ that are needed for the calculation of μ_i^d, $i = 1, ..., 8^d$. Fig. 6 shows the complete 2D op-

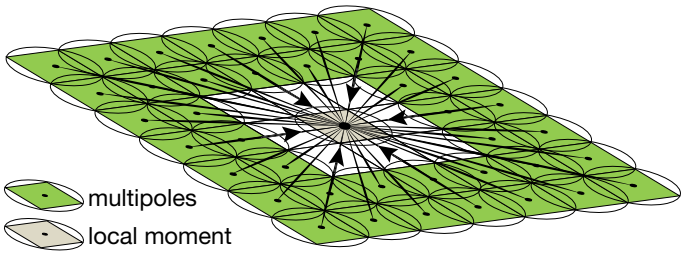


Figure 6: 2D representation of the operator set $\mathcal{M}$. In 3D, there are 342 possible positions (green) relative to the central box (light brown). Since the nearest neighbors (white) and self interactions are excluded, the number of active operators reduces to 316.

erator set $\mathcal{M}$ for the `well separation criterion` $w = 1$. For each $d = 1, ..., \mathcal{D}$ the set $\mathcal{M}^d$ requires storing 343 pointers. A unique mapping function $\mathtt{opindex}(x, y, z)$ returns a 1D index for each $\mathbf{M}_j^d \in \mathcal{M}^d$, $j = 0, ..., 343 - 1$, $d = 1, ..., \mathcal{D}$ with

$$x + 3 + (y + 3)\delta + (z + 3)\delta^2, \tag{20}$$

where $x = x_\omega - x_\mu$, $y = y_\omega - y_\mu$, $z = z_\omega - z_\mu$ are the relative positions of ω and μ in 3D and $\delta = 3 + 4\sigma$. Here, σ denotes the number of directly interacting box layers according to the `well separation criterion`. In PBC, any $\omega_j^d \in \mathcal{L}_i^d$ with an out-of-box position is remapped from the corresponding periodic position with the `periodic_remapping`$(\cdot)$ function. Listing 3 shows a basic implementation of the

Listing 2: The loops that start the M2L operators in the octree traverse the whole tree, compute the M2L interaction set and launch one M2L translation for each interaction in the computed set (pseudocode).

```
1 //chosen tree depth D
2 //for each relevant depth
3 int d,muZ,muY,muX,muXp,muYp,muZp
4 for(int d = 1; d <= D; ++d)
5 {
6  for(muZ=0; muZ<std::pow(2, d); ++muZ)
7  {
8   for(muY=0; muY<std::pow(2, d); ++muY)
9   {
10   for(muX=0; muX<std::pow(2, d); ++muX)
11   {
12    int muI = index(muX, muY, muZ, d);
13    //parent box of mu
14    muXp = muX/2;
15    muYp = muY/2;
16    muZp = muZ/2;
17    int omZ, omY, omX;
18    //operator index
19    int opI;
20    //computation of interaction lists
21    //based on the parent box information
22    for(omZ=(muZp-1)*2; omZ<(muZp+2)*2; ++omZ)
23    {
24     for(omY=(muYp-1)*2; omY<(muYp+2)*2; ++omY)
25     {
26      for(omX=(muXp-1)*2; omX<(muXp+2)*2; ++omX)
27      {
28       opI = opindex(omX-muX, omY-muY, omZ-muZ);
29       //remap out of bounds indices
30       periodic_remapping(omX, omY, omZ);
31       omegaI    = index(omX, omY, omZ, d);
32       M2L(mu[muI], omega[omegaI], M[opI]);
```

`M2L`$(\cdot)$ function, which computes Eq. (15) up to order p in four nested for-loops.

3.3 Three CUDA parallelization approaches

The previous section described the basic sequential FMM. We will now present three different parallelization approaches. Approach (i) is conceptually straight-

Listing 3: Basic implementation of the M2L operato (pseudocode).

```
1 typedef triangular_matrix Tm;
2 void M2L(Tm* mu, Tm* omega, Tm* M)
3 {
4  for (int l = 0; l <= p; ++l)
5  {
6   for (int m = 0; m <= l; ++m)
7   {
8    for (int j = 0; j <= p; ++j)
9    {
10    for (int k = -j; k < j; ++k)
11    {
12     mu(l,m) += M(j+l,k+m) * omega(j,k);
```

forward, nevertheless it achieves decent speedups compared to a sequential CPU implementation with only minor parallelization work. It directly maps for-loops to CUDA threads, leaving the sequential program structure nearly unmodified. Approach (ii) performs well for high accuracy demands (high multipole orders $p \gtrsim 12$, double precision), however it scales poorly for smaller p. Approach (iii) minimizes the number of arithmetic operations by exploiting the symmetry of the $\mathbf{M}$ operator. It scales well in the broad range $0 \leq p \leq 20$, however it requires additional data structures to minimize bookkeeping and to utilize symmetries.

3.3.1 Naïve parallelization approach (1)

The complete M2L operation in 3D requires 11 loops as shown in Listing 2. Listing 4 shows the comparison of the FMM loop structure and its naïve CUDA parallelization counterpart. Since CUDA provides a 3-component vector `threadIdx` to control the parallel execution of the threads, the main idea is to map the loops directly to the CUDA structures. To this end, we use a transformation between 1D and 3D indices. Any sequence of n indices $i = 0, .., n-1$ can be transformed into n m-dimensional tuples of indices $(x_0, ..., x_{m-1})$ with $x_j = (i/m^j) \bmod m,\ j = 0, ..., m-1$. As our FMM operates on cubic domains, the number of boxes is $\texttt{dim}(d)$ in each orthogonal direction for depths $d = 1, .., \mathcal{D}$. The loop over the M2L interaction set $\mathcal{L}$ is of a fixed size $(6 \times 6 \times 6)$ on each depth d. The M2L operation contains four for-loops of size $\leq p$.

Listing 4: Direct mapping of FMM octree and M2L loops (top part, lines 1–20) to CUDA threads (bottom part, lines 22–43) (pseudocode).

```
1 *** M2L loops structure ***
2 //loop over depths
3 int d, muZ, muY, muX, omZ, omY,omX
4 for (int d = 1; d <= D; ++d)
5  //three loops over tree boxes
6  for(muZ = 0; muZ < std::pow(2, d); ++muZ)
7   for(muY = 0; muY < std::pow(2, d); ++muY)
8    for(muX = 0; muX < std::pow(2, d); ++muX)
9     //computation of the M2L interaction lists
10    for(omZ=(muZ/2-1)*2;omZ<(muZ/2+2)*2;++omZ)
11     for(omY=(muY/2-1)*2;omY<(muY/2+2)*2;++omY)
12      for(omX=(muX/2-1)*2;omX<(muX/2+2)*2;++omX)
13       //M2L operation
14       int l,m,j,k;
15       for(l = 0; l <= p; ++l)
16        for(m = 0; m <= l; ++m)
17         for(j = 0; j <= p; ++j)
18          for(k = -j; k < j; ++k)
19           //one complex multiplication
20           //and addition
21
22 *** CUDA M2L structure ***
23 //loop over tree levels d = 1,..,D on CPU
24 int dim = std::pow(2,d)
25 int p1 = p+1
26 //computation of the one-dimensional index i
27 int i = blockIdx.x * blockDim.x + threadIdx.x
28 //three loops over tree boxes on level d
29 int muZ = (i/(p1*p1*p1*216*dim*dim))%dim
30 int muY = (i/(p1*p1*p1*216*dim))%dim
31 int muX = (i/(p1*p1*p1*216))%dim
32 //computation of the M2L interaction lists
33 int om_z = (i/(p1*p1*p1*6*6))%6 - (muZ/2-1)*2
34 int om_y = (i/(p1*p1*p1*6))%6 - (muY/2-1)*2
35 int om_x = (i/(p1*p1*p1))%6 - (muX/2-1)*2
36 //M2L operation
37 int l = (i/(p1*p1))%p1
38 int m = (i/p1)%p1
39 if(m > l)
40   return;
41 int j = i%p1
42 for (int k = -j; k < j; ++k)
43 //one complex multiplication and addition
```

The iteration over the tree levels is performed by the CPU. Since the M2L operations are level independent, the kernels are spawned asymmetrically for each level of the octree enabling overlapped execution. The last for-loop in Listing 3 is performed

sequentially by each thread. It accumulates partial sums

$$\mu_{lm}(j) = \sum_{k=-j}^{j} \mathbf{M}_{l+j,m+k}\omega_{jk}, \tag{21}$$

of the complete dot product

$$\mu_{lm} = \sum_{j=0}^{p}\sum_{k=-j}^{j} \mathbf{M}_{l+j,m+k}\omega_{jk} = \sum_{j=0}^{p} \mu_{lm}(j). \tag{22}$$

This reduces the number of atomic writes by a factor of $\mathcal{O}(p)$.

The naïve strategy allows a rapid FMM parallelization. Replacing the existing serial FMM loops with the corresponding CUDA index calculations leads to speedups that make the FMM algorithm applicable for moderate problem sizes. No additional data structures and code modification are required. However, the achieved bandwidth and parallelization efficiency is still far from optimal on the tested hardware.

3.3.2 CUDA dynamic parallelism approach (2)

A substantial performance issue of the naïve approach is integer calculation, which introduces a significant overhead even for large p. For $d = 1, ..., \mathcal{D}$, $p^3 \times 216 \times 8^d$ threads are started, where each computes a valid pair of 3D source and target box indices to perform $\mathcal{O}(p)$ complex multiplications and additions $\mu_{lm} = \mu_{lm} + \mathbf{M}_{l+j,m+k}\omega_{jk}$. This leads to $\mathcal{O}(p^3)$ redundant source and target box index computations. A possible mitigation of the expensive index computations is Dynamic Parallelism [30]. It allows to spawn kernels recursively, what simplifies hierarchical calculations. The dynamic approach exploits Dynamic Parallelism to avoid the expensive bookkeeping calculations of the naïve approach.

The determination of μ_i^d for $i = 0, ..., 8^d$ and $d = 1, ..., \mathcal{D}$ is done on the host as given in Listing 1. To this aim, the octree is traversed in 3D to precompute the coordinates (x_μ, y_μ, z_μ) of μ_i^d and the origin coordinates $(x_\mathcal{L}, y_\mathcal{L}, z_\mathcal{L})$ of $\mathcal{L}_i^d$. Together with 1D index i of μ_i^d, they are passed as arguments to a parent kernel spawned for each μ_i^d. Let $\mathcal{P}_J^d := \{j_0, ..., j_7\}$ be the set of indices of all boxes at depth d contained in the parent box of ω_j^d. Thus, for an arbitrary μ_i^d it holds: $\mathcal{L}_i^d = \bigcup_{J=0}^{25} \mathcal{P}_J^d$, $\mathcal{P}_J^d \cap \mathcal{P}_{J'}^d = \emptyset$, for any distinct pair $J \neq J'$. Listing 5 shows the parent kernel, that is engaged only in octree operations. To better utilize concurrency, it is started with $B_x = B_y = B_z = 3$ for $6 \times 6 \times 6$ interaction sets $\mathcal{L}_i^d$. The parent kernel consists of threads that can be uniquely identified with (tid_x, tid_y, tid_z) tuples. Each thread precomputes one 3D source positions $(x_\omega, y_\omega, z_\omega)$ of $\omega_{j_0}^d$, $j_0 \in \mathcal{P}_J^d$, $J = 0, ..., 25$ (lines 4–6). The index j_0 of the proper operator $\mathbf{M}_{j_0 \to i}^d$ is calculated from the relative 3D coordinates of μ_i^d and $\omega_{j_0}^d$ (line 12). Since the parent box index I of μ_i^d contains only its direct neighbors ($\mathcal{P}_I^d \nsubseteq \mathcal{L}_i^d$), the direct neighbors in $\mathcal{P}_I^d$ are omitted. Fig. 7 illustrates the dynamic kernel. Each parent kernel spawns 26 child kernels with

Listing 5: Parent kernel in the dynamic approach. It determinates valid ω coordinates and spawns child kernels performing M2L computations (pseudocode).

```
1 parent_kernel(int muI, int muX, int muY,
2                int muZ, int xL, int yL, int zL,...)
3 {
4   int omZ = threadIdx.z * 2 + zL;
5   int omY = threadIdx.y * 2 + yL;
6   int omX = threadIdx.x * 2 + xL;
7
8   if(muZ/2 != omZ/2 ||
9      muY/2 != omY/2 ||
10     muX/2 != omX/2 )
11  {
12    //operator index
13    int opI;
14    opI = opindex(omX-muX, omY-muY, omZ-muZ);
15    periodic_remapping(omX, omY, omZ);
16    int omegaI = index(omX, omY, omZ);
17
18    dim3 block(p+1,p+2,1);
19    dim3 grid(2,2,2);
20    child_kernel(muI, omegaI, opI,...);
21  }
22 }
```

$G_x = G_y = G_z = 2$ and 2D `blocks` $B_x = p+1, B_y = p+2, B_z = 1$. One child kernel computes eight $\mathcal{O}(p^4)$ M2L transformations between one target μ_i^d and all ω_j^d, $j \in \mathcal{P}_J^d$.

Listing 6 shows child kernel computations, which can be divided in two parts. In the first part, ω_j^d, $j \in \mathcal{P}_J^d$ and the operator $\mathbf{M}_{j\to i}^d$ are determined. Since indices of ω and $\mathbf{M}$ are provided by the parent kernel, the $(2 \times 2 \times 2)$ grid facilitates a straightforward way to determine eight different ω_j^d, $j \in \mathcal{P}_J^d$ with $j = j' + Bid_x + Bid_y * \texttt{dim}(d) + Bid_z * \texttt{dim}(d)^2$, where j' is the index passed by the parent kernel. The operators $\mathbf{M}_j^d$ are obtained correspondingly, by replacing $\texttt{dim}(d)$ with 7 and j' with the operator index passed by the parent kernel.

To decrease the number of global memory accesses, `shared memory` is used to cache ω and $\mathbf{M}$. This is advantageous, since one M2L operation executes $\mathcal{O}(p^4)$ steps on $\mathcal{O}(p^2)$ data structures. The triangular shaped matrices are converted to 1D arrays in `shared memory`, allowing consecutive addressing in the for-loops performing the reduction step. The `shared memory` storage index s_i of each moment $\omega_{lm} \in \omega_j^d$ is calculated with $s_i = l^2 + l + m$, where $l := tid_y$ and $m := tid_x$. A similar approach holds for the operator $\mathbf{M}_j^d$, however, since $\mathbf{M} \in \mathcal{O}(2p^2)$, threads need to be reused to write the elements into `shared memory`.

In our implementation, the direct neighbor operator is given the size $p = 0$. This

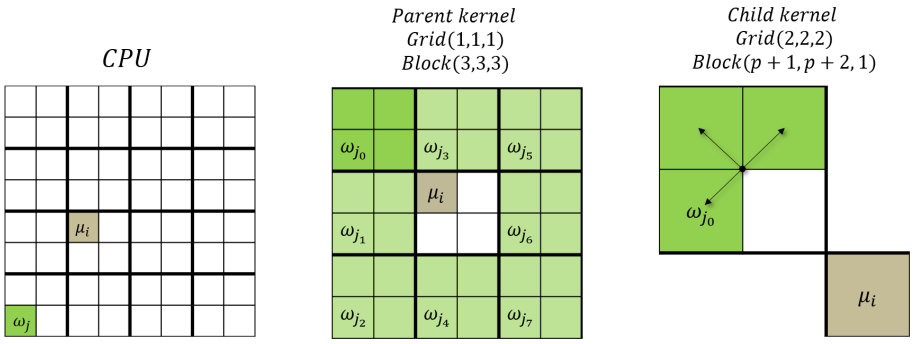


Figure 7: Dynamic M2L scheme. The CPU computes a μ_i^d and the corresponding $\mathcal{L}_i^d$. The parent kernel determines all valid $\mathcal{P}_J^d \subsetneq \mathcal{L}_i^d$ sets, whereas the child kernel performs the M2L operations for $\omega_{j_k}^d$, $j_k \in \mathcal{P}_J^d$, $k = 0, ..., 7$ and the target μ_i^d.

allows to skip the remaining nearest neighbor interactions of μ_i^d for $\mathcal{P}_J^d \subsetneq \mathcal{L}_i^d$ by checking for the condition $p = 0$.

In the second part, the `j_k_reduction()` function computes the M2L operation. To mitigate the waste of threads due to the triangular shape of the μ, ω and $\mathbf{M}$ and to minimize the number of atomic global memory writes, each thread executes the two innermost loops, Eq. (22), sequentially. However, a straightforward approach leads to `warp divergence`, since threads that correspond to $m > l$ indices of the target moments need to be skipped. Splitting the innermost loop, such that it is partly performed by threads $m > l$ circumvents this issue. Fig. 8 and Listing 7 show a possible splitting scheme of the M2L operation. It uses $(p + 1) * (p + 2)$ threads, where some unique thread pairs are mapped to the same target index tuple (l, m) of a target element $\mu_{lm} \in \mu_i^d$. These compute a distinct part of the reduction.

Listing 6: Child kernel in the dynamic approach (pseudocode).

```
template<typename M2L,typename OMEGA,
         typename MU>
void child_kernel(int muI, int omIstart,
                  int opIstart,
                  int opSize,
                  M2L** M_Operator,
                  OMEGA** Omega,
                  MU** Mu)
{
  int boxX = blockIdx.x;
  int boxY = blockIdx.y;
  int boxZ = blockIdx.z;

  extern __shared__ complex_type cache[];
  complex_type* shared_M=cache;
  complex_type* shared_O=&cache[opSize];
  //M2L operator
  int opI = opIstart + boxZ*7*7 + boxY*7 + boxX;
  M2L* M = M_Operator[opI];
  //skipping nearest neighbor operations
  if(M->p() == 0)
    return;
  //omega index
  int omI;
  omI = omIstart + boxZ * dim(d) * dim(d)
      + boxY * dim(d) + boxX;
  OMEGA* O = Omega[omI];

  int m = threadIdx.x;
  int l = threadIdx.y;
  int tx = l*(p+1) + m;

  if(tx < (p+1)*(p+1))
  {
    //writing of the moments and operators
    //intto shared memory
    shared_O[l*l+l+m] = O(l,m);
    shared_M[l*l+l+m] = M(l,m);
  }
  __syncthreads();

  MU* mu = Mu[muI];
  j_k_reduce(shared_O, shared_M, l, m, mu);
}
```

Listing 7: Reduction function. It computes new target indices ll, mm from arguments l, m in a way that innermost loop is split to be performed by all threads in the block (pseudocode).

```
void j_k_reduce(complex_type& shared_O,
                complex_type& shared_M,
                int l, int m, MU* mu)
{
  int l1, t1, ll, mm, f, f1, k_start, k_end;
  t1 = tx - l;
  l1 = t1/p_out;
  //f = 0,1
  f  = 1 - l - l1;
  f1 = 1 - f;
  ll = f1 * l1 + f * m;
  mm = f1 * m  + f * l1;

  complex_type mu_l_m = 0.0;
  complex_type mu_l_m_j, op, om;
  for (int j = 0; j <= p; j++)
  {
    mu_l_m_j = 0.0;

    int jl       = j + ll;
    int jl2_jl_m = jl * jl + jl + mm;
    int jj2_j    = j * j + j;

    for (int k = f * j; k <= f * j + j + f; k++)
    {
        int lk = k - j;
        op = shared_M[jl2_jl_m + lk]);
        om = shared_O[jj2_j    + lk]);
        mu_l_m_j += op * om;
    }
    //changing sign in the odd j-th element
    mu_l_m += change_sign_if_odd_j(j, mu_l_m_j);
  }
  //atomic add on global memory
  *mu(ll, mm) += mu_l_m;
}
```

The described dynamic approach allows for further optimization, as it splits the computation in two independent parts. The parent kernels handle the octree position evaluation, whereas the child kernels implement the M2L computation. The efficiency of the this approach is satisfactory for high multipole orders. The necessity of an efficient parallelization also for small p leads to the next approach.

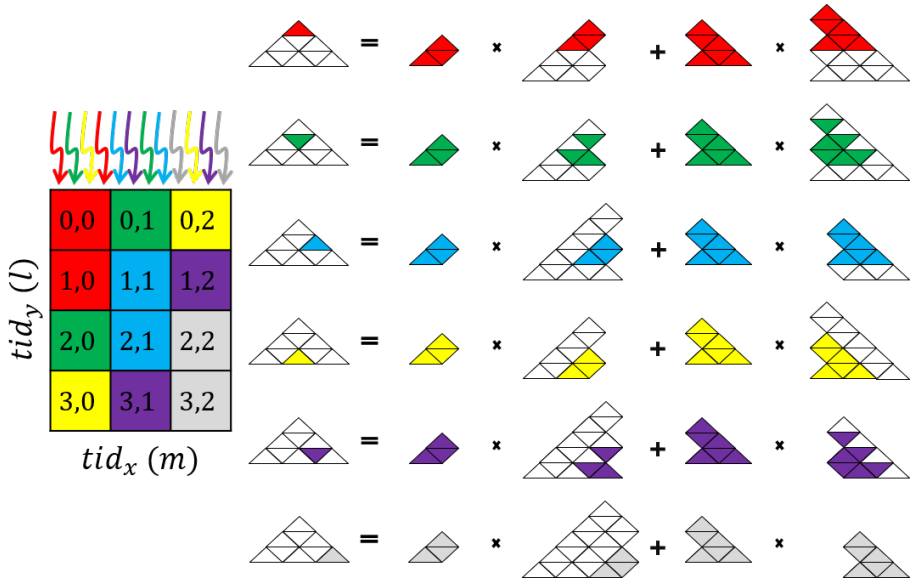


Figure 8: Thread splitting scheme to minimize `warp divergence`. Example of 12 threads performing six reductions. Threads are allocated in a 2D block. Each target (l, m) is shared by two threads (same color) performing a different part of the dot product.

3.3.3 Presorted list-based approach with symmetric operators (3)

In this approach, the FMM interaction pattern is precomputed for higher efficiency. Additionally, operator symmetries are exploited to reduce both the number of complex multiplications as well as global memory access.

Octree interactions precomputation

The pattern of interactions between the octree boxes is static, hence it can be precomputed and stored. This step does not need to be performance-optimal, as it is done only once at the start of a simulation that typically spans millions of time steps. In a PBC octree configuration, for each ω_i^d, $d = 1, ..., \mathcal{D}$, $i = 0, ..., 8^d - 1$ there exists an interaction set $\mathcal{R}_i^d$, $|\mathcal{R}_i^d| = 208$. It consists of all the indices j of local moments μ_j^d that a multipole ω_i^d is contributing to. Note that the index sets $\mathcal{R}_i^d$ and $\mathcal{L}_i^d$ (defined in Section *The Multipole-to-Local (M2L) transformation*) are identical. For higher efficiency, the sets $\mathcal{R}_i^d$ are precalculated and stored as lists $\hat{\mathcal{R}}_i^d = (j_0, ..., j_{188})$ (with nearest neighbors skipped) with an arbitrary but fixed order. In addition, the corresponding operators are determined and stored as lists $\hat{\mathcal{M}}_i^d$ ordered in a way that

$$\mu_{j_k} = \mathbf{M}_{j_k} \omega_i, \quad k = 0, ..., 188 \tag{23}$$

describes all valid M2L transformations of the i-th multipole moment. The precalculation of $\hat{\mathcal{R}}_i^d$ and $\hat{\mathcal{M}}_i^d$ is achieved by sequentially traversing the octree as shown in Listings 1–2. Within the `omega` class, each ω_i stores 189 pointers to its targets μ_{j_k} and the corresponding pointers to M2L operators. We make sure that the internal list orders preserve the validity of Eq. (23), so that it suffices to store direct pointers to the target moments and operators instead of their indices. We will use the index list notation $\hat{\mathcal{R}}_i^d$ and $\hat{\mathcal{M}}_i^d$, keeping in mind that the lists actually store pointers.

The precomputed interaction lists $\hat{\mathcal{R}}_i^d$ and $\hat{\mathcal{M}}_i^d$ enable the following kernel configuration. The number of distinct M2L transformations for each ω_i^d is set with $G_z = 189$. $G_x = G_y = p$, thus $\mathcal{O}(p^2)$ CUDA `blocks` are spawned to handle $\mathcal{O}(p^4)$ interactions. The remaining $\mathcal{O}(p^2)$ operations are executed sequentially by each thread. $B_x = 8^{d-1}$ is the number of boxes on octree level $d-1$. This value fits CUDA architecture requirements for the `blocksize` particularly well, as it is always an even multiple of `warpsize`. For $d > 4$, B_x exceeds the the largest allowed `blocksize` of 1024, so we replicate kernel launches for consecutive ω in strides of size 1024.

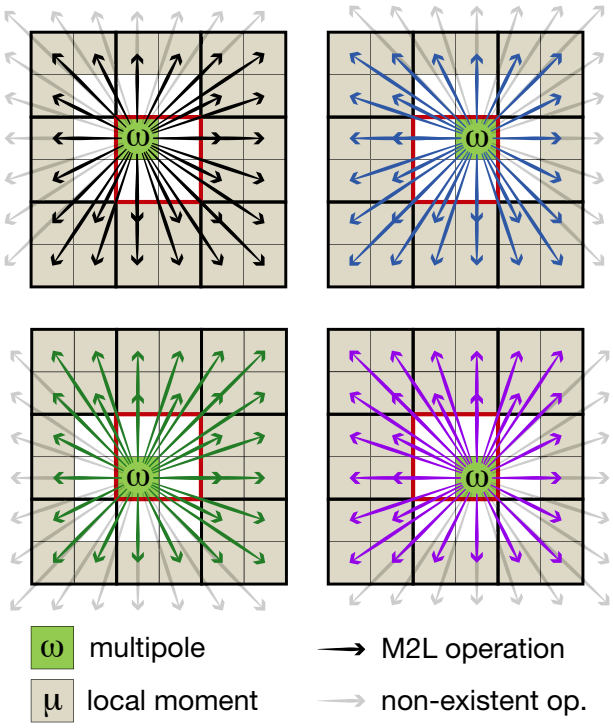


Figure 9: Different operator groups. The groups are represented by arrows of distinct color, depend on the position of ω within its parent (red squares). Four possible 2D operator groups $\mathcal{G}_s$ are shown. In 3D, there are eight different operator groups.

Fig. 9 illustrates that for each position of ω_i^d within its parent box a specific interaction set $\mathcal{R}_i^d$ results. Therefore, the precomputed lists $\hat{\mathcal{R}}_i^d$ and $\hat{\mathcal{M}}_i^d$ require reshuffling to facilitate the straightforward indexing within the kernel. Let $\mathcal{G}_s$, $s =$

$0, ..., 7$ denote the eight possible groups governed by the position of ω within the parent box. The 8^d pointers to ω are reshuffled such that eight consecutive sequences of 8^{d-1} pointers in memory belong to the same group $\mathcal{G}_s$. Rigorously, for ω_i^d where $d = 1, ..., \mathcal{D}$ and $i = 0, ..., 8^d$ it holds $\omega_{i_k}^d \in \mathcal{G}_s, k = s8^{d-1}, ..., (s+1)8^{d-1}-1, s = 0, ..., 7$.

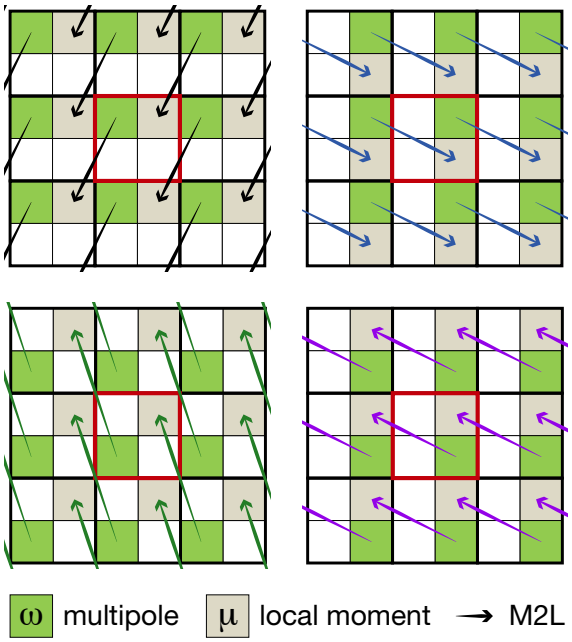


Figure 10: Parallelization of M2L operations for the operator groups shown in Fig. 9. Each single operator is processed in parallel for all boxes on a level by starting one CUDA block for each $\omega_i \in \mathcal{G}_s$ (arrows of same color show one example for each operator group). The kernels are replicated for $s = 0, ..., 7$.

With reshuffled ω pointers the CUDA parallelization proceeds as shown in Fig. 10. One kernel is started for each $\mathcal{G}_s$, $s = 0, ..., 7$. Each thread tid_x within a `block` evaluates one dot product (Eq. (22)). The pointers to the targets $\mu_{j_k}^d$ and to $\mathbf{M}_{j_k}^d$ are accessed with precomputed and presorted lists $\hat{\mathcal{R}}_i^d$ and $\hat{\mathcal{M}}_i^d$ without any additional integer operation, hence $Bid_z \equiv j_k$. The particular moments $(\mu_{lm})_{j_k}^d$, $j_k \in \mathcal{G}_s$ are evaluated in a parallel CUDA `block` with $l = Bid_x$ and $m = Bid_y$. As these are block variables, skipping of the $m > l$ part does not lead to `warp divergence`. Additionally, only the relevant part of the triangular operator matrix (Fig. 5) needs to be loaded into `shared memory` to be accessed by all threads tid_x within the `block`.

A further improvement of the kernel is gained by rearranging the moments in memory. Threads tid_x of an l, m `block` access the same moments ω_{lm} of consecutive ω_i, with $i = tid_x$. For `warpwise` coalesced memory access, the arrangement of the moments in memory is switched from Array of Structures (AoS) to Structure of Arrays (SoA). The moments $(\omega_{lm})_i^d$, $i = 0, ..., 8^d - 1$ are stored in SoA triangular matrices such that for fixed l, m, the i indexed elements are contiguous in memory.

Operator symmetry

The symmetry of associated Legendre polynomials

$$P_{lm}(-x) = (-1)^{l+m} P_{lm}(x) \tag{24}$$

emerges directly from their definition (Eqs.5–6). It allows to reduce the size of the operator set $\mathcal{M}^d$, as shown in Fig. 11. In 3D, the complete operator set spans a

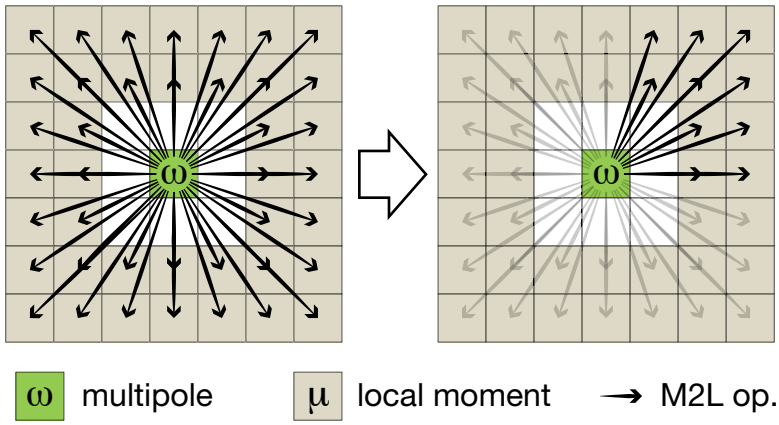


Figure 11: Reduction of the M2L operator set. The complete operator set $\mathcal{M}^d$ (left) and a reduced operator set $\tilde{\mathcal{M}}^d \subseteq \mathcal{M}^d$ (right) in 2D. Each black arrow symbolizes one M2L operator.

cube with the operators originating from its center to all $7^3 - 3^3$ subcubes. The reduced operator $\tilde{\mathcal{M}}^d$ contains 56 M2L operators $\omega_i \to \mu_{j_x}$ $(x = 0, ..., 55)$ of one of the octants. Let the octant of the cube with parameters $\theta, \phi \in [0, \frac{1}{2}\pi]$ in spherical coordinates be the reference octant. The generation of particular operator moments with symmetrical functions

$$\mathbf{M}_{lm} = \frac{(l-m)!}{\|x\|_2^{l+1}} P_{lm}(\cos\theta) e^{im\phi}, \tag{25}$$

where

$$e^{im\phi} = \cos(m\phi) + i\sin(m\phi) \tag{26}$$

yields three operator symmetry groups containing orthogonal operators that differ only by their sign. Fig. 12 shows the symmetry groups in $\tilde{\mathcal{M}}^d$.

Depending on the relative position of ω_i in its parent box, the interaction set $\mathcal{R}_i$ requires a different subset of operators in $\mathcal{M}$, see Fig. 9. Hence, for each $\mathcal{G}_s$, $s = 0, ..., 7$ on each depth $d = 0, ..., \mathcal{D}$, the operator set $\mathcal{M}^d$ and the corresponding

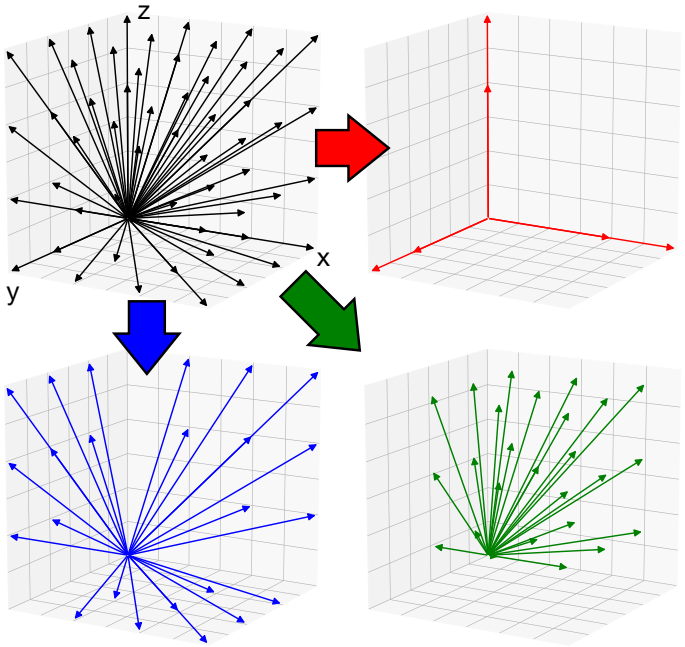


Figure 12: Grouping of the M2L operators according to their symmetry properties. The reduced operator set as shown in black in the upper left panel (a 2D version is shown in the right panel of Fig. 11) is sorted into three groups (red, blue, green) depending on whether the operator is aligned with an axis (red), or within one of the xy, xz, or yz planes (blue), or none of that (green). Each of the red operators (in x, y, and z direction) has one symmetrical counterpart (in $-x$, $-y$, and $-z$ direction, respectively). Each of the blue operators has four symmetrical counterparts each (one in each quadrant of the plane). Each of the remaining operators has eight symmetrical counterparts each (one in each octant of the cube).

index set $I = \{0, ...188\}$ can be split in disjunct subsets $\mathcal{T}$ such that

$$\begin{aligned}
\mathcal{T}_{\alpha_1} &= \{\mathbf{M}_j \,|\, \nexists\, \mathbf{M}_i \in \mathcal{M} : \mathrm{abs}(\mathbf{M}_i) \\
&\qquad = \mathrm{abs}(\mathbf{M}_j)\ \forall j \in I\ \} \\
\mathcal{T}_{\alpha_2} &= \{\mathbf{M}_{i_0}, \mathbf{M}_{i_1} \,|\, \mathrm{abs}(\mathbf{M}_{i_x}) \\
&\qquad = \mathrm{abs}(\mathbf{M}_{i_y})\ \forall x, y\ \in \{0, 1\}\} \\
\mathcal{T}_{\alpha_3} &= \{\mathbf{M}_{i_0}, ..., \mathbf{M}_{i_3} \,|\, \mathrm{abs}(\mathbf{M}_{i_x}) \\
&\qquad = \mathrm{abs}(\mathbf{M}_{i_y})\ \forall x, y\ \in \{0, 1, 2, 3\}\} \\
\mathcal{T}_{\alpha_4} &= \{\mathbf{M}_{i_0}, ..., \mathbf{M}_{i_7} \,|\, \mathrm{abs}(\mathbf{M}_{i_x}) \\
&\qquad = \mathrm{abs}(\mathbf{M}_{i_y})\ \forall x, y\ \in \{0, 1, 2, 3, 4, 5, 6, 7\}\}
\end{aligned} \tag{27}$$

where $\mathrm{abs}(X) := \mathrm{abs}(X_{l,m}), l = 0, ..., p, m = -l, ..., l$ and $\alpha_1 = (0, ..., 6)$, $\alpha_2 = (7, ..., 27)$, $\alpha_3 = (28, ..., 48)$, $\alpha_4 = (49, ..., 55)$. This property allows to reduce the $G_z = 189$ to $G_z = 56$, however further kernel modifications are required.

To make efficient usage of the operator symmetry, the lists $\hat{\mathcal{M}}_i^d$ for each ω_i are

again reordered such that

$$\hat{\mathcal{M}}_i^d = (\mathcal{T}_{\alpha_1}, \mathcal{T}_{\alpha_2}, \mathcal{T}_{\alpha_3}, \mathcal{T}_{\alpha_4}). \tag{28}$$

The corresponding lists $\hat{\mathcal{R}}_i$ need to be resorted as well, to preserve Eq. (23). A bitset $\mathcal{B}$ is added to the $\mathbf{M}$ operator class to store signs of its elements. As these are complex values, it takes two bits to store the signs. The bitset is indexed in an array like manner with the most significant bit as the zero-th element. With $u = 2(l^2+l)+2m$, $\mathcal{B}(u)$ and $\mathcal{B}(u+1)$ represents the sign of the real and complex part of $\mathbf{M}_{lm}$, respectively. Since bitsets are precomputed during the operator initialization phase, they do not introduce any performance degradation whereas their additional memory footprint is negligible. Fig. 13 shows an example of an M2L computation with bitsets. A single operator access from global memory computes 1, 2, 4, or 8 target moments μ depending on the operator type $\mathcal{T}$ as given in Eq. (27). The

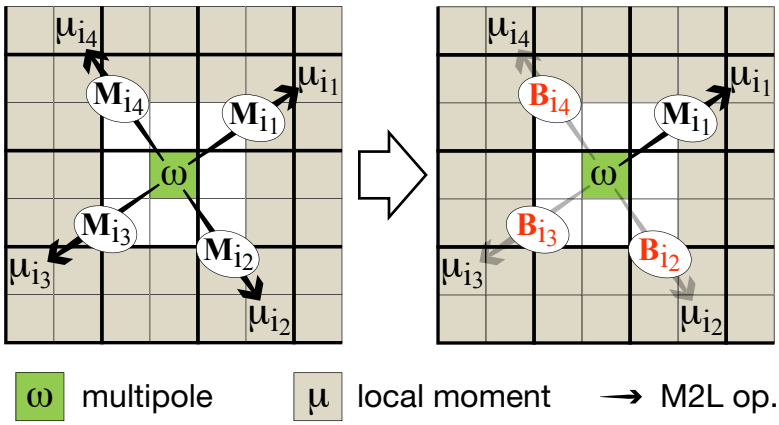


Figure 13: M2L operator symmetry exploitation. Left: Computation of four M2L operations with four orthogonal operators $\mathbf{M}$ (black arrows). Right: The use of bitsets $\mathbf{B}$ (orange) minimizes the redundant memory accesses and reduces the number of complex multiplications.

target moments μ_{t_γ}, with $\gamma = 0, ..., \beta$, $\beta = 1, 2, 4, 8$ for any source ω are computed as follows. The intermediate products $\mu_{lm,jk} := \mathbf{M}_{l+j,m+k}\ \omega_{jk}$ of the complete dot product Eq. (22) are split in

$$\begin{aligned} ac &= \Re(\mathbf{M}_{l+j,m+k})\Re(\omega_{jk}) \\ bd &= \Im(\mathbf{M}_{l+j,m+k})\Im(\omega_{jk}) \\ ad &= \Re(\mathbf{M}_{l+j,m+k})\Im(\omega_{jk}) \\ bc &= \Im(\mathbf{M}_{l+j,m+k})\Re(\omega_{jk}) \end{aligned} \tag{29}$$

where $\mathbf{M}_{l+j,m+k}$ are the elements of the reference operator $\mathbf{M}_{t_0}$. The split products change their signs for $(\mu_{lm,jk})_{t_\gamma}$, $\gamma \neq 0$ according to $\hat{\mathcal{B}}_{t_\gamma} = \mathcal{B}_{t_\gamma} \oplus \mathcal{B}_{t_0}$, where $\mathcal{B}_{t_0}$ is the bitset of the reference operator, $\oplus$ is the binary XOR operator and $\gamma = 0, ..., \beta$, $\beta = 0, 2, 4, 8$ depending on the operator symmetry group $\mathcal{T}_{\alpha_x}$, $x = 1, 2, 3, 4$. For $x = 1$ there is no symmetric counterpart of the operator $\mathbf{M}$. For $x > 1$ the intermediate

moments calculation is

$$\begin{aligned}(\mu_{lm,jk})_{t_\gamma} = &\operatorname{sign}(ac, \hat{\mathcal{B}}_{t_\gamma}, u) - \operatorname{sign}(bd, \hat{\mathcal{B}}_{t_\gamma}, u+1) \\ &+i(\operatorname{sign}(ad, \hat{\mathcal{B}}_{t_\gamma}, u) + \operatorname{sign}(bc, \hat{\mathcal{B}}_{t_\gamma}, u+1)),\end{aligned} \tag{30}$$

with

$$\operatorname{sign}(x, \hat{\mathcal{B}}, u) = \begin{cases} x, \text{ if } \hat{\mathcal{B}}(u) = 0 \\ -x, \text{ if } \hat{\mathcal{B}}(u) = 1. \end{cases} \tag{31}$$

The `sign` function changes the sign of x by shifting the u-th bit of the bitset $\hat{\mathcal{B}}$ to the left most bit position and by evaluating the $x \oplus \hat{\mathcal{B}}_{shifted}$ subsequently. This creates no `warp divergence` since the sign change is a result of the arithmetic and logical operations.

The constant size of the lists, see Eq. (28), allows to implement the M2L kernel for the symmetry groups $\mathcal{T}_{\alpha_x}$, $x = 1, 2, 3, 4$ as a function template, resulting in a single kernel that efficiently treats different groups $\mathcal{T}_{\alpha_x}$. Listing 8 shows the kernel configuration for different symmetry groups. For different operator groups $\mathcal{G}_s$, $s = 0, ..., 7$, the kernels are replicated. The computation of the index i of ω_i is straightforward as pointers to ω_i are contiguous for any $\mathcal{G}_s$. For each symmetry group $\mathcal{T}_{\alpha_x}$, $x = 1, 2, 3, 4$ one kernel with distinct template parameters is started, where the first parameter describes the cumulative offset of a particular $\mathcal{T}_{\alpha_x}$ in $\hat{\mathcal{M}}_i^d$ and the second one is the number of symmetrical operators within the current $\mathcal{T}_{\alpha_x}$. The size of α_x, $x = 1, 2, 3, 4$ is set to G_z. The kernels are launched for all configurations $\mathcal{T}_{\alpha_x} \times \mathcal{G}_s$ asymmetrically to utilize concurrency. Listing 9 shows the implementation of the symmetric kernel.

Listing 8: Configuration and launches of the symmetric M2L kernel (pseudocode).

```
#define STREAMS 32
dim3 b(boxes_on_this_depth/8,1,1);
//multipoleorder p
dim3 g1(p,p,7);
dim3 g2(p,p,21);
dim3 g3(p,p,21);
dim3 g4(p,p,7);
int k = 0;
for (int s = 0; s < 8; ++s)
{
  M2L_symmetric<0,1>
  <<<g1,b,sm_size,stream[++k%STREAMS]>>>(s,...);
  M2L_symmetric<7,2>
  <<<g2,b,sm_size,stream[++k%STREAMS]>>>(s,...);
  M2L_symmetric<49,4>
  <<<g3,b,sm_size,stream[++k%STREAMS]>>>(s,...);
  M2L_symmetric<133,8>
  <<<g4,b,sm_size,stream[++k%STREAMS]>>>(s,...);
}
```

At the beginning, the reference operator $\mathbf{M}_{t_0}$ and the bitsets of all orthogonal operators $\mathcal{B}_{t_\gamma}$ are loaded into `shared memory`. Depending on the group $\mathcal{T}_{\alpha_x}$, $x = 1, 2, 3, 4$ different number of bitsets is loaded. The `if` statement, that tests the value of the template parameter `group_type`, is resolved at compile time. The second part of Listing 9 shows the split complex multiplication implementation. The double nested for-loop computes $1, 2, 4$ or 8 $(\mu_{lm})_{t_\gamma}$ depending on the symmetry group $\mathcal{T}_{\alpha_x}$. The `if` statement within the innermost loop is resolved at compile time as well.

Listing 9: The symmetrical M2L kernel (pseudocode).

```
template<typename offset, typename group_type,
         typename Real>
M2L_symmetric(int s, int p, int d, Operators* B,
              Omegas* Omega){

 int l     = blockIdx.x;
 int m     = blockIdx.y;
 int B_id  = offset + blockIdx.z * group_index;
 int omega_offset = s*blockDim.x + nb(d-1);
 //shared memory alloc
 unsigned int* bits0,bits1,bits2,bits3,
               bits4,bitsb5,bits6,bits7;
 complex* B0;
 //global memory accesses and shared memory writes
 if(threadIdx.x == 0)
   bits0 = B[B_id]->bitset;
   for(int j = 0; j <= p; ++j)
     for(int k = -j; k <= j; ++k)
       B0[(j*(j+1) + k] = B[B_id](l+j,m+k);
   if(group_type > 1)
     bits1 = B_list[B_id+1]->bitset^bits0;
   if(group_type > 2)
     bits2 = B_list[B_id+2]->bitset^bits0;
     bits3 = B_list[B_id+3]->bitset^bits0;
   if(group_type > 4)
     bits2 = B_list[B_id+4]->bitset^bits0;
     bits3 = B_list[B_id+5]->bitset^bits0;
     bits2 = B_list[B_id+6]->bitset^bits0;
     bits3 = B_list[B_id+7]->bitset^bits0;
 Real ac,bd,ad,bc;
 complex mu_lm0, mu_lm1, mu_lm2,...
 for(int j = 0; j <= p; ++j)
   for(int k = -j; k <= j; ++k)
     int u = 2*j*(j+1) + 2*k;
     O_jk=Omega(j,k,omega_offset+threadIdx.x);
     B_jk=B0[(j*(j+1) + k];
     ac = O_jk.real*B_jk.real;
     bd = O_jk.imag*B_jk.imag;
     ad = O_jk.real*B_jk.imag;
     bc = O_jk.imag*B_jk.real;
     mu_lm0 += complex(ac-bd,ad+bc);
     if(group_type > 1)
       mu_lm1 += complex(sign(ac,bits1,u)-sign(bd,bits1,u+1)
                         ,
                         sign(ad,bits1,u)+sign(bc,bits1,u+1)
                 );
     if(group_type > 2)
       mu_lm2 += complex(sign(ac,bits2,u)-sign(bd,bits2,u+1)
                         ,
                         sign(ad,bits2,u)+sign(bc,bits2,u+1)
                 );
       mu_lm3 += ...
     if(group_type > 4)
       mu_lm4 += ...
       mu_lm5 += ...
       mu_lm6 += ...
       mu_lm7 += ...
}
```

4 Benchmarks and discussion

We will now benchmark the performance and analyze the scaling behavior of the three different parallelization approaches described above.

4.1 General FMM scaling behavior

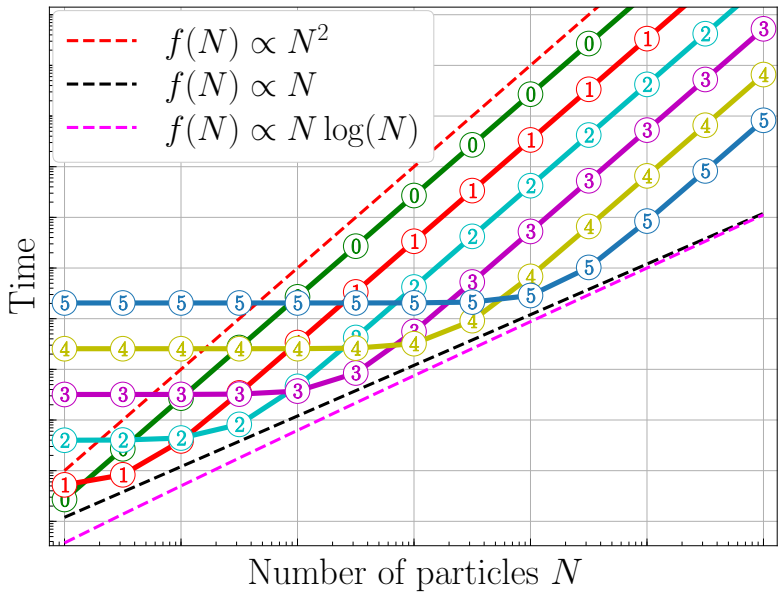


Figure 14: Qualitative sketch of the FMM scaling behavior. The optimal linear scaling (black dashes) with particle number N is achieved if and only if the tree depth $\mathcal{D}$ (as indicated by the colored numbers) is properly chosen for each N. For a constant $\mathcal{D}$, for small N, FMM run time is dominated by the far field computations, whereas for growing N, ultimately $\mathcal{O}(N^2)$ scaling results (red dashes).

Fig. 14 sketches the FMM scaling behavior with respect to the number of particles N, which is $\mathcal{O}(N)$ when the tree depth $\mathcal{D}$ is chosen properly. However, locally the FMM scales like $\mathcal{O}(n^2)$, with n being the average number of particles in the boxes at the lowest level $\mathcal{D}$. For a fixed multipole order p, at constant depth, a fixed number of $\mathcal{O}(p^4)$ far field operations are performed. In the regime of small N, the $\mathcal{O}(p^4)$ far field part completely dominates the FMM runtime, which is therefore essentially independent of N. At some critical N, the scaling curve switches to a quadratic behavior, because the P2P computations start to dominate the overall runtime. To benefit from the optimal linear scaling for growing N, the depth needs to be chosen properly. Varying p affects the slope of the overall linear scaling.

4.2 Benchmarking procedure

All performance tests were executed on a workstation with an Intel Xeon CPU E5-1620@3.60 GHz with 16 GB physical memory and a Pascal NVIDIA GeForce GTX 1080 Ti with 3584 CUDA Cores. This GPU has a theoretical single precision peak performance of 11.6 TFLOPS and maximal bandwidth of 484 GB/s. The device code was compiled with `NVCC` 9.1. All kernel timings were measured with the help of `cudaEvent`s and represent the average runtime of 100 runs.

In our performance comparisons we focus on $\mathcal{D} = 3$, as it provides sufficient parallelism to get proper performance metrics, which are also valid for $\mathcal{D} = 4$. For higher depths, the computation requires more kernel spawns due to limitations of `blocksize`, which leads to performance decrease. On the tested GTX 1080 Ti GPU a depth of $\mathcal{D} = 3$ is suitable for particle counts of $4 \times 10^4 - 3 \times 10^5$, whereas higher N requires $\mathcal{D} = 4$ and $\mathcal{D} = 5$ for optimum performance. The current implementation of the symmetric parallelization approach allows for a maximum depth of $\mathcal{D} = 5$ at which up to $N \approx 1.2 \times 10^6$ particles can be handled efficiently. The limitation is caused by memory optimization, in which redundant pointers are stored to minimize the costs of scattered global memory writes. It can be switched off allowing for $\mathcal{D} = 6$ and system sizes up to $N \approx 10^8$. On the tested Pascal GPU this optimization increases performance by about 10%, while on a Turing GPU the effect of the optimization is negligible [32].

4.3 Microbenchmarking

To evaluate the different parallelization approaches in context of the underlying hardware, we estimated the GPU performance bounds for the M2L transformation operation. To this aim, we implemented two benchmarking microkernels, which execute exactly the number of arithmetic operations and memory accesses as the M2L operation does. However, additional possible performance bottlenecks [41] like `warp divergence`, non coalesced memory accesses, `shared memory` bank conflicts and atomic writes are eliminated. The microkernels were then used to determine the effective runtime bounds for our three different parallelization approaches.

Fig. 15 shows the absolute runtimes of the microkernels. To get the maximal theoretical throughput of the M2L kernel, we assumed the execution of three global memory accesses, eight bytes each, to perform one complex multiplication and one global addition, i.e. $\mathcal{O}(p^4)$ memory accesses for $\mathcal{O}(p^4)$ arithmetic operations. This results in a clearly memory bound kernel with $\mathcal{O}(p^4)$ scaling in the examined range of p. For the lower bound we assumed an idealized scenario: for each box in the octree, $\mathcal{O}(p^2)$ memory accesses are performed for the moments and operators, whereas the data needed for $\mathcal{O}(p^4)$ operations is assumed to be available in registers. The full $\mathcal{O}(p^4)$ memory access approach utilizes nearly the full bandwidth of the GPU, achieving 370 Gb/s. However, the computation performance is only about 89 GFLOPS, which is $\approx 0.8\%$ of the GPU's peak performance. The second, $\mathcal{O}(p^2)$ memory access

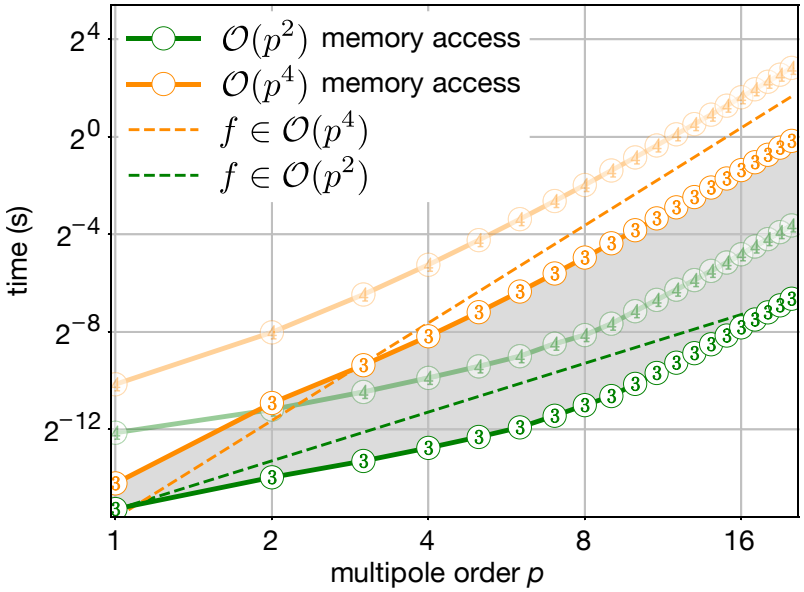


Figure 15: Runtime of the memory-bound microkernel (orange) and of the compute-bound microkernel (green) for two tree depths $\mathcal{D} = 3$ and $\mathcal{D} = 4$, as indicated by the encircled numbers. The run times of the implemented M2L kernels are expected in the shaded area between memory-bound and compute-bound microkernels.

approach, varies depending on p. For $p < 10$ we observe subquadratic scaling of the execution time, indicating that the $\mathcal{O}(p^4)$ arithmetic operations are fully hidden. For $p > 10$ the curve shifts to the $\mathcal{O}(p^4)$ regime, achieving up to 8 TFLOPS, i.e. 70% of the GPU's peak performance. Compute and memory utilization are balanced at $p = 10$, which is where the curve switches from $\mathcal{O}(p^2)$ to the $\mathcal{O}(p^4)$ slope.

4.4 Performance comparison

We will now discuss the efficiency of the three proposed parallelization approaches.

4.4.1 Naïve kernel

Fig. 16 shows the absolute executions times of the different kernels for $\mathcal{D} = 3$ and $\mathcal{D} = 4$. In the whole p range, the naïve kernel's theoretical arithmetic intensity (see Roofline model [57]), is a lot smaller than the ratio $R = 23.95$ FLOPS/byte obtained from the GPU's FLOP rate (11.6 TFLOPS) divided by its memory transfer rate (484 GB/s). This indicates that the kernel is bandwidth limited. However, additional integer computation is required for the calculation of 3D octree indices of the interaction sets $\mathcal{L}$. Fig. 18a shows that for the naïve kernel, much less than 10% of the issued instructions are useful floating point operations. A large computational

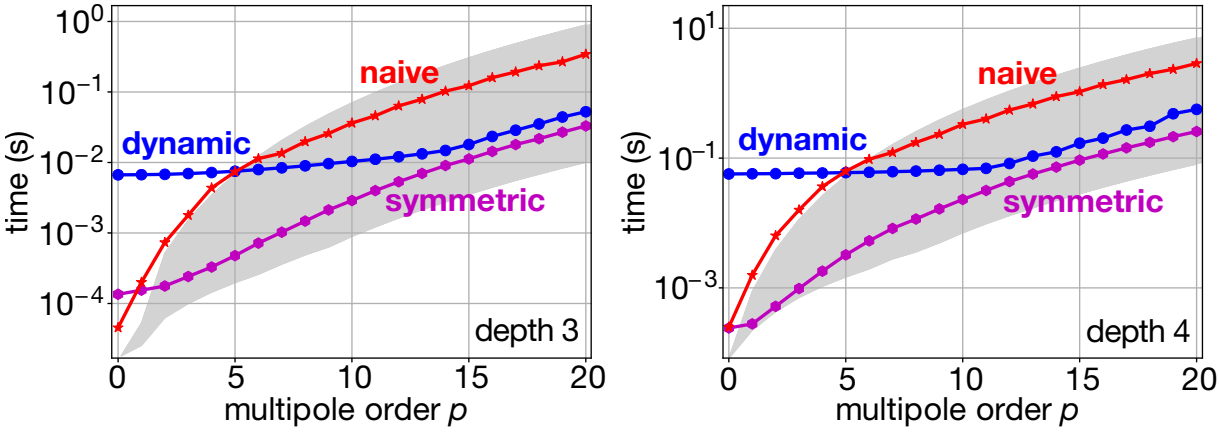


Figure 16: Runtime comparison of the three different M2L implementations. For each multipole order, $8^{\mathcal{D}} \times 189$ single M2L operations are computed for $\mathcal{D} = 3$ (left) and $\mathcal{D} = 4$ (right). The gray area marks the gap between the memory bound and the compute bound reference microkernel shown in Fig. 15.

overhead emerges from a high number of integer operations in the innermost for-loop. Here, each of $\mathcal{O}(p)$ complex multiplications requires 31 integer additions, 16 integer multiplications, nine modulo operations and 11 integer divisions. In addition, performance is significantly reduced by `warp divergence`, since different threads in a `block` resolve the condition $m > l$ (line 36 of Listing 4). This effect is labeled as `no operation` in Fig. 18a. Avoiding `warp divergence` would require different kernels for each $0 \leq p \leq 20$, since mapping of indices to threads differs at each p. Fig. 18b shows, how well the GPU is utilized by the naïve kernel. As both memory and compute achieve roughly 50% of maximal possible utilization, the performance is likely limited by the latency of arithmetic or memory operations.

The maximum number of achieved FLOPS (p=19) is at 2% of the GPU capability, see Fig. 17 (left). The effective bandwidth reaches nearly 500 GB/s, which is more than the maximum memory throughput of the GPU. As seen in Fig. 16, the naïve kernel achieves runtimes similar to the memory bound microkernel. For values $p > 6$, the kernel is slightly faster than the memory-bound reference kernel, and that is for the following reason. The fact that the innermost for-loop of the naïve kernel is executed sequentially allows cache reuse. Each element ω_{lm} can be reused 189 times for a different M2L transformation and each element of the operator $\mathbf{M}$ is reused 8^d times at tree depth d. This leads to local cache throughput of roughly 3,500 GB/s, which approaches the maximal theoretically possible cache bandwidth of the GPU.

Additionally, the achieved occupancy per each Streaming Multiprocessor (SM) is at 46% of a possible maximum of 50% at this kernel configuration, a limit caused by the number of registers (64) used in the kernel.

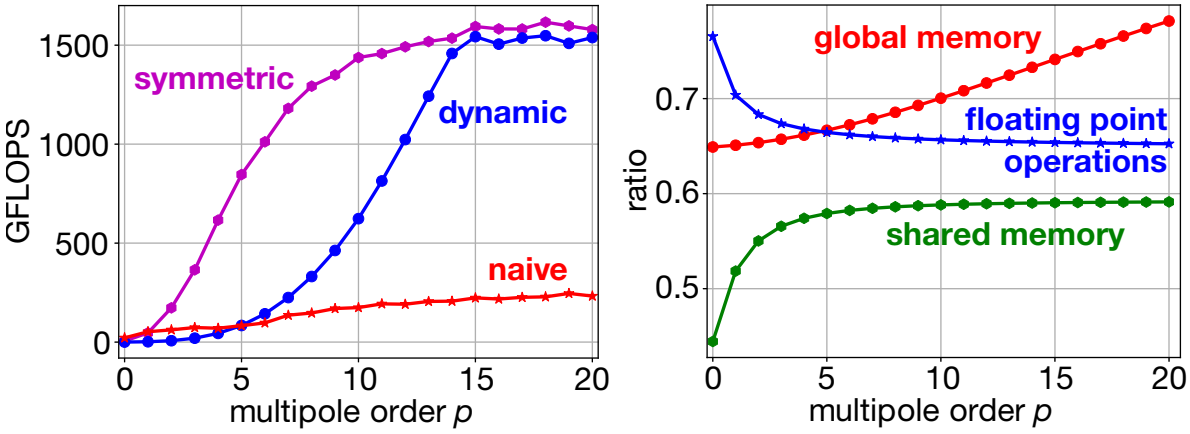


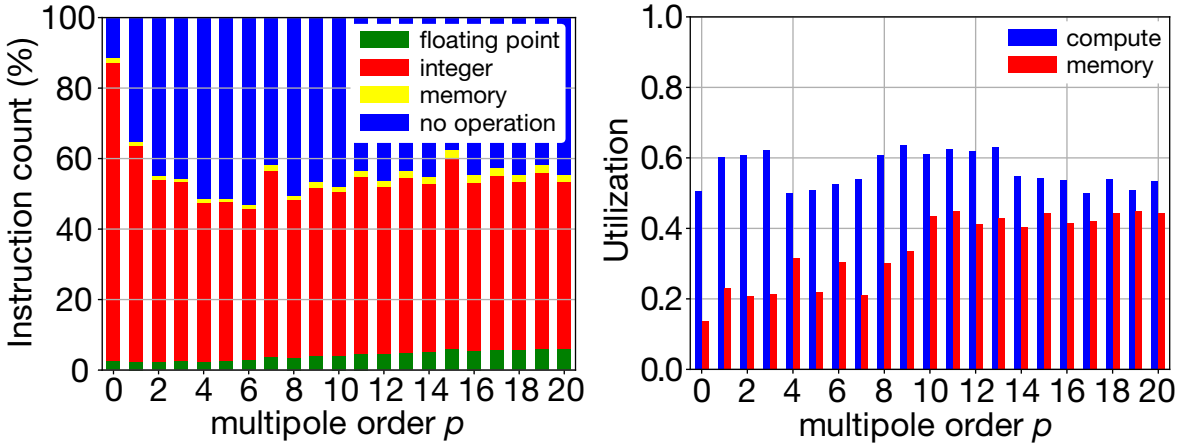
Figure 17: M2L kernel performance comparison by two different metrics. Left: FLOPS achieved by different kernels. Right: Ratio of global and shared memory accesses and floating point operations for the symmetric kernel with respect to the non-symmetric approach.

4.4.2 Dynamic kernel

This kernel utilizes Dynamic Parallelism to minimize the index overhead computation introduced by the naïve kernel. The sizes of the child kernels $(2, 2, 2)$ allow for utilization of concurrency on the GPU, since the work in the child kernels is fully independent. On the underlying hardware the maximum number of resident grids per device is limited to 32.

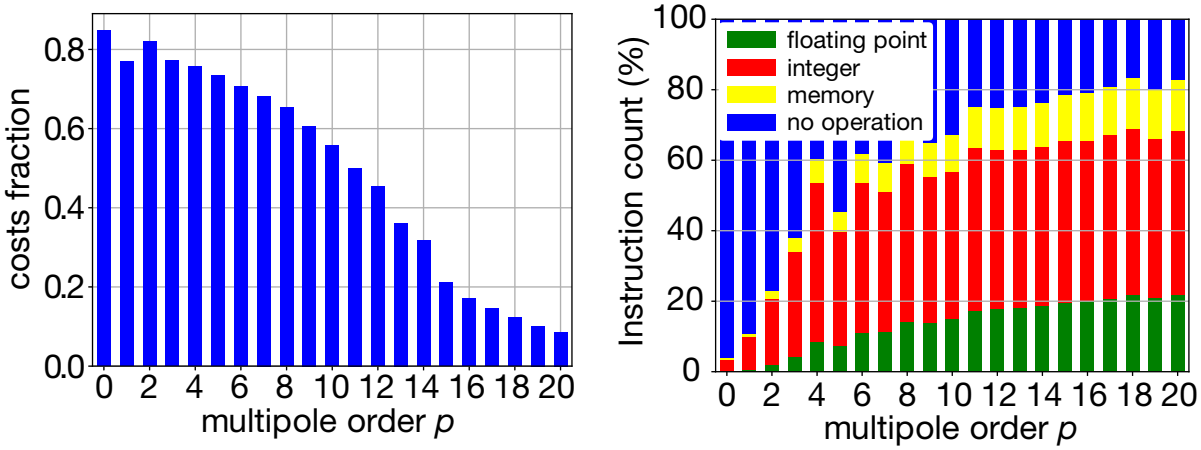
Fig. 19a shows the relative costs emerging from launching the child kernels, which become irrelevant only for large p. At small p, the latencies dominate the computation time, leading to an almost constant runtime for $p \leq 10$ that can be seen in Fig. 16 for the dynamic kernel. Fig. 19b shows the instruction distribution for the dynamic kernel. From $p \approx 3$ on, the fraction of floating point operations is significantly larger than for the naïve kernel. However, the large number of integer operations and `warp divergence` still limits the performance. Another issue is the small `block` size of the child kernels, which limits the SM occupancy for different p values. For $p < 5$, e.g., each `block` consists of only one `warp`. This limits the SM utilization, as 32 `blocks` but 64 `warps` can be executed simultaneously. For $p \geq 5$, the occupancy is only limited by the register usage, achieving nearly 100% of the theoretical possible occupancy. Limiting the register usage, however, increases the local memory traffic and does not further enhance the performance.

As `shared memory` usage is an essential part of the dynamic kernel, we tested how its utilization affects the overall performance. Fig. 20a and Fig. 20b show the `shared memory` throughput and the GPU utilization of the dynamic kernel, respectively. For $p < 12$ the kernel is clearly compute-bound, hence the `shared memory` operations are fully hidden. At $p = 12$–15 we can observe a balance between memory and compute operations. `Shared memory` is limiting only for $p > 15$, however



(a) Instructions distribution. (b) Memory and compute utilization.

Figure 18: Performance analysis of the naïve M2L kernel.



(a) Kernel launching latencies. (b) Instructions distribution.

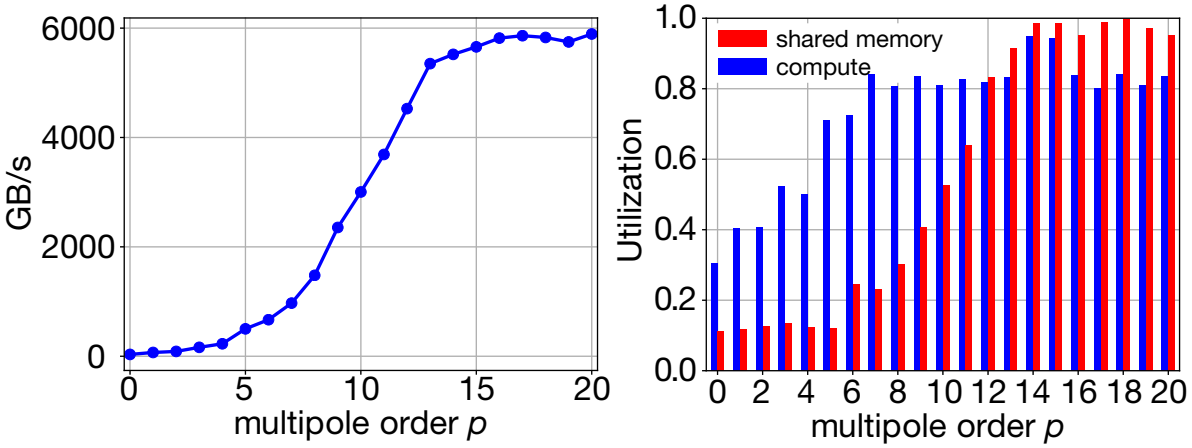
Figure 19: Performance analysis of the dynamic M2L kernel.

the achieved throughput of 6,000 GB/s is at the limit of the underlying GPU.

The overall performance of the kernel gets considerably higher compared to the naïve kernel for $p > 6$, achieving a maximum of 1,600 GFLOPS for $p = 20$, see Fig. 17 (left). Nevertheless, memory and FLOPS peaks are achieved only for $p > 15$. At $p < 5$, the dynamic kernel performs worse than the $\mathcal{O}(p^4)$ benchmarking microkernel mainly due to kernel launch latencies mentioned above.

4.4.3 Symmetric kernel

The symmetric M2L kernel is composed of four subkernels started asynchronously for each symmetry group $\mathcal{T}_{\alpha_x}$, $x = 2, 3, 4$ (see Eq. (27)). Fig. 21 shows the achieved speedup compared to the standard implementation. Additionally, it also shows the speedups of each symmetric part $\mathcal{T}_{\alpha_x}$. As expected, the 8-way symmetric kernel



(a) Shared memory throughput.

(b) Shared memory and compute utilization.

Figure 20: Performance analysis of the dynamic M2L kernel.

performs best, followed by the 4-way and then the 2-way symmetric kernel. The overall speedup of the symmetric kernel combines the speedups of the subkernels. However, the achieved overall speedup is not directly proportional to the number of the symmetrical counterparts, because the additional bit shifting and sign changing operations introduce a growing overhead for larger symmetry. In addition, register utilization is larger for the kernels with higher symmetry, harming the achieved SM occupancy. From here on, we will combine the metrics for all subkernels, referring

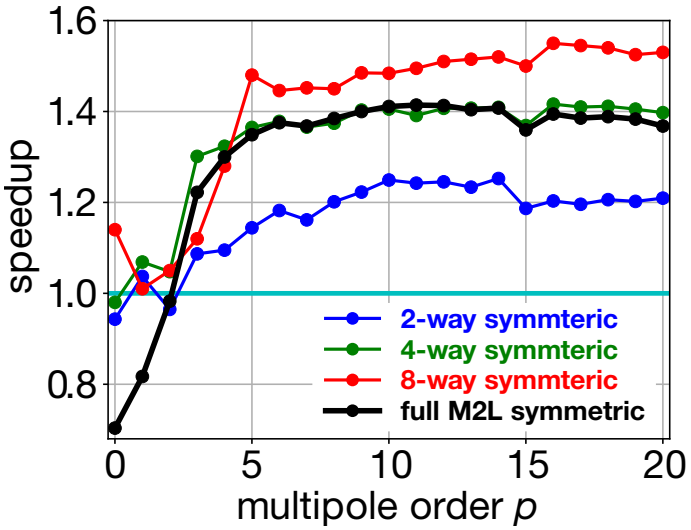
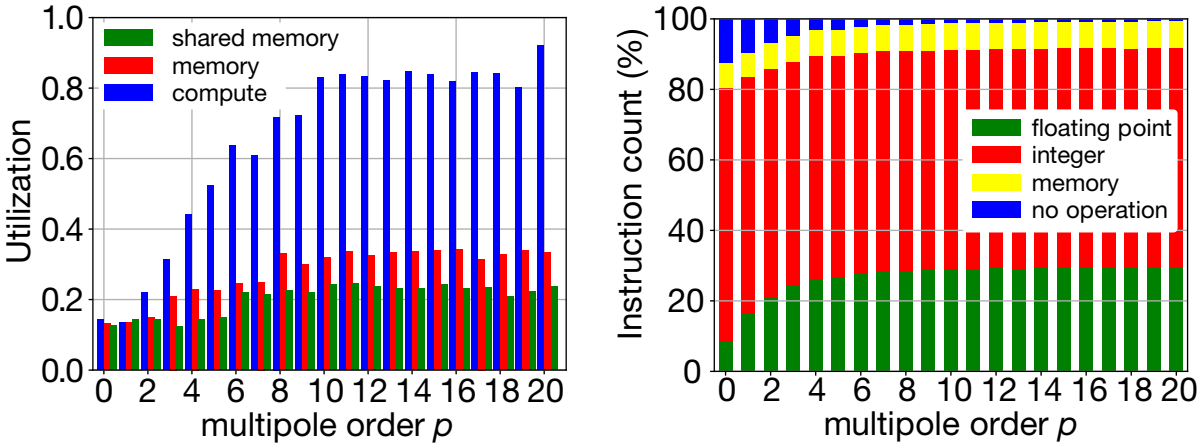


Figure 21: Speedups due to symmetry properties for different symmetry groups $\mathcal{T}_{\alpha_x}$, $x = 2, 3, 4$ (colored) and for the whole symmetric M2L operator kernel (black) compared to a non-symmetric implementation (cyan).

to them as one symmetric kernel. These metrics take into account the overlapping execution of the symmetric subkernels.



(a) Hardware utilization.

(b) Instructions distribution.

Figure 22: Performance analysis of the symmetric M2L kernel.

Fig. 22a shows how the hardware is utilized by the symmetric kernel. This kernel is compute-bound over nearly the complete p range. As seen in Fig. 22b, the fraction of floating point operations is significantly larger than in both previous approaches. `Warp divergence` is eliminated completely. The `no operation` part, resulting from pipeline latencies becomes negligible for $p > 6$. In this range, the `gridsizes` $((p+1)*(p+2)/2, 7, 1)$ and $((p+1)*(p+2)/2, 21, 1)$ of particular symmetric subkernels are too small to provide enough `blocks` to utilize the complete device, so that pipeline latencies become an issue. However, with larger p hardware utilization increases.

The register usage of the symmetric subkernels varies between 48% and 71% (not shown). Based on the kernel configuration, the theoretical maximum possible average SM occupancy for all subkernels is 57%. The kernels achieve 50% and are mainly limited by register usage. Further occupancy optimization is unlikely to further increase performance markedly, as kernels with a large register usage do not require optimal occupancy [55].

As Fig. 17 (left) shows, the FLOP rate of the symmetric kernel is much higher in the range $1 < p < 16$ compared to the other two kernels. However, the FLOP rates achieved by the symmetric and dynamical kernel for $p \geq 15$ are similar. Nevertheless, the symmetric scheme clearly outperforms the dynamic one when comparing the absolute execution times, see Fig. 16. Fig. 17 (right) demonstrates that, compared to the non-symmetric kernel, the symmetric kernel needs fewer floating point operations for the M2L stage.

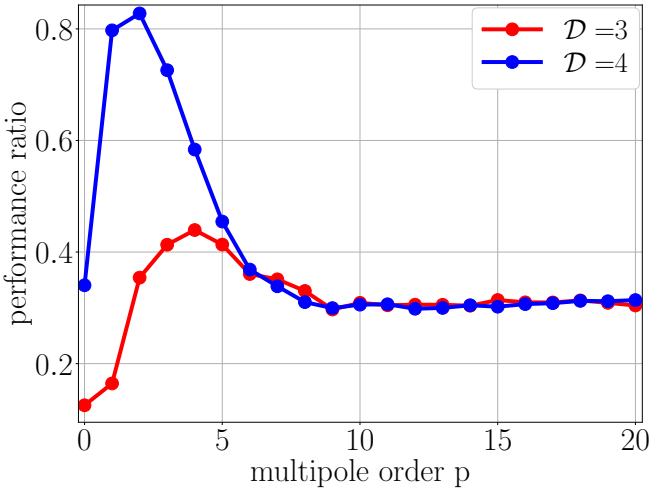


Figure 23: Absolute performance ratio of the symmetric M2L kernel with respect to the compute-bound reference microkernel with $\mathcal{O}(p^2)$ memory loads for $\mathcal{D} = 3$ and $\mathcal{D} = 4$.

Fig. 23 shows the absolute performance ratio of the symmetric kernel compared to the compute-bound reference microkernel. It achieves roughly 30% of the reference microkernel performance in $2 < p < 21$. Additionally, for both depths, the kernel shows an ideal scaling as the showed ratio remains nearly constant for $p > 5$. Hence, the scaling of the symmetric kernel follows the compute-bound reference microkernel scaling. It achieves a subquadratic scaling for $p < 10$ and switches slowly to $\mathcal{O}(p^4)$ regime, that is limited only by arithmetic throughput, compare Fig. 15 and Fig. 16.

5 Conclusions

Here, we have presented three different CUDA parallelization approaches for the Multipole-to-Local operator, which is performance limiting for the overall FMM performance. The first approach preserves the sequential loop structure and does not require any special data structures. It makes use of CUDA Unified Memory to achieve decent speedups compared to a sequential CPU implementation. It is useful, e.g., for rapid prototyping or for simulation systems with small to moderate numbers of particles. However, it comes with a large computational overhead due to additional integer operations.

The second approach, which exploits CUDA Dynamic Parallelism, avoids this drawback and achieves very good performance at high accuracy demands, i.e. for large multipole orders. Its main drawback is a lack of performance at low multipole orders, for which the first scheme performs better.

The third approach uses abstractions of the underlying octree and interaction pat-

terns to allow for enhanced, efficient GPU utilization and it exploits the symmetries of the Multipole-to-Local operator. As a result, it scales perfectly with growing multipole order p maintaining a very good performance in the whole benchmarked multipole range ($0 < p < 20$)

Our FMM implementation has been optimized for biomolecular simulations and has been incorporated into GROMACS as an alternative to the established PME Coulomb solver [32]. We anticipate that, thanks to the inherently parallel structure of the FMM, future multi-node multi-GPU implementations will eventually overcome the PME scaling bottlenecks [9, 35, 36].

Acknowledgements

This project was supported by the DFG priority programme *Software for Exascale Computing* (SPP 1648). A special thanks goes to Jiri Kraus from NVIDIA who supported this project in the early stage of its development and to R. Thomas Ullmann who took part in writing the FMM-GROMACS interface and unit tests.

References

[1] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, pages 19–25, 2015.

[2] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based FMM for heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, 28(9):2608–2629, 2016.

[3] M. Allen and D. Tildesley. Computer simulation of liquids. 1987. *New York: Oxford*, 385, 1989.

[4] Y. Andoh, N. Yoshii, K. Fujimoto, K. Mizutani, H. Kojima, A. Yamada, S. Okazaki, K. Kawaguchi, H. Nagao, K. Iwahashi, F. Mizutani, K. Minami, S. Ichikawa, H. Komatsu, S. Ishizuki, Y. Takeda, and M. Fukushima. MODYLAS: A Highly Parallelized General-Purpose Molecular Dynamics Simulation Program for Large-Scale Systems with Long-Range Forces Calculated by Fast Multipole Method (FMM) and Highly Scalable Fine-Grained New Parallel Processing Algorithms. *J. Chem. Theory Comput.*, 9(7):3201–3209, 2013.

[5] Y. Andoh, N. Yoshii, and S. Okazaki. Extension of the fast multipole method for the rectangular cells with an anisotropic partition tree structure. *J. Comput. Chem.*, pages 1–15, 2020.

[6] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann,

M. Pippig, D. Potts, and G. Sutmann. Comparison of scalable fast methods for long-range interactions. *Phys. Rev. E*, 88:063308, 2013.

[7] P. Blanchard, B. Bramas, O. Coulaud, E. Darve, L. Dupuy, A. Etcheverry, and G. Sylvand. ScalFMM: A Generic Parallel Fast Multipole Library. In *SIAM Conference on Computational Science and Engineering (SIAM CSE 2015)*, Salt Lake City, United States, 2015.

[8] J. A. Board, J. W. Causey, J. F. Leathrum, A. Windemuth, and K. Schulten. Accelerated molecular dynamics simulation with the parallel fast multipole algorithm. *Chem. Phys. Lett.*, 198(1):89–94, 1992.

[9] J. A. Board, C. W. Humphres, C. G. Lambert, W. T. Rankin, and A. Y. Toukmaji. Ewald and Multipole Methods for Periodic N-Body Problems. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. E. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 459–471. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[10] L. Bock, C. Blau, G. Schröder, I. Davydov, N. Fischer, H. Stark, M. Rodnina, A. Vaiana, and H. Grubmüller. Energy barriers and driving forces in tRNA translocation through the ribosome. *Nat. Struct. Mol. Biol.*, 20:1390–1396, 2013.

[11] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *J. Comput. Phys.*, 155(2):468–498, 1999.

[12] H. Dachsel. An error-controlled fast multipole method. *J. Chem. Phys.*, 132: 119901, 2010.

[13] J. M. Dawson. Particle simulation of plasmas. *Rev. Mod. Phys.*, 55:403–447, Apr 1983.

[14] H. Ding, N. Karasawa, and W. A. Goddard. Atomic level simulations on a million particles: The cell multipole method for Coulomb and London nonbond interactions. *J. Chem. Phys.*, 97(6):4309–4315, 1992.

[15] H.-Q. Ding, N. Karasawa, and W. A. Goddard. The reduced cell multipole method for Coulomb interactions in periodic systems with million-atom unit cells. *Chem. Phys. Lett.*, 196(1):6–10, 1992.

[16] R. O. Dror, R. M. Dirks, J. Grossman, H. Xu, and D. E. Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annu. Rev. Biophys.*, 41:429–452, 2012.

[17] M. Eichinger, H. Grubmller, H. Heller, and P. Tavan. FAMUSAMM: An algorithm for rapid evaluation of electrostatic interactions in molecular dynamics simulations. *J. Comput. Chem.*, 18(14):1729–1749, 1997.

[18] N. Engheta, W. D. Murphy, V. Rokhlin, and M. S. Vassiliou. The fast multipole method (FMM) for electromagnetic scattering problems. *IEEE Transactions on Antennas and Propagation*, 40(6):634–641, June 1992.

[19] U. Essmann, L. Perera, M. Berkowitz, T. Darden, and H. Lee. A smooth particle mesh Ewald method. *J. Chem. Phys.*, 1995.

[20] W. Fong and E. Darve. The black-box fast multipole method. *J. Comput. Phys.*, 228(23):8712–8725, 2009.

[21] A. G. Garcia, A. Beckmann, and I. Kabadshow. *Accelerating an FMM-Based Coulomb Solver with GPUs*, pages 485–504. Springer International Publishing, Berlin, Heidelberg, 2016.

[22] N. Y. Gnedin. Hierarchical particle mesh: An FFT-accelerated fast multipole method. *Astrophys. J. Suppl. Ser.*, 243(2):19, jul 2019.

[23] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325 – 348, 1987.

[24] L. Greengard and V. Rokhlin. A new version of the Fast Multipole Method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269, 1997.

[25] N. A. Gumerov and R. Duraiswami. FMM accelerated BEM for 3D Laplace & Helmholtz equations. In *Proceedings Int. Conf. on Boundary Element Techniques*, 2006.

[26] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *J. Comput. Phys.*, 227(18):8290 – 8313, 2008.

[27] T. Hansson, C. Oostenbrink, and W. van Gunsteren. Molecular dynamics simulations. *Curr. Opin. Struct. Biol.*, 12(2):190–196, 2002.

[28] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *J. Chem. Theory Comput.*, 2008.

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

[30] S. Jones. Introduction to dynamic parallelism. In *GPU Technology Conference Presentation*, volume 338, 2012.

[31] M. Knap and P. Czarnul. Performance evaluation of Unified Memory with prefetching and oversubscription for selected parallel CUDA applications on NVIDIA Pascal and Volta GPUs. *The Journal of Supercomputing*, 75(11): 7625–7645, Nov 2019.

[32] B. Kohnke, C. Kutzner, and H. Grubmüller. A GPU-accelerated Fast Multipole Method for GROMACS: performance and accuracy. *Manuscript submitted to J. Chem. Theory Comput.*, 2020.

[33] B. Kohnke, T. R. Ullmann, A. Beckmann, I. Kabadshow, D. Haensel, L. Morgenstern, P. Dobrev, G. Groenhof, C. Kutzner, B. Hess, H. Dachsel, and H. Grubmüller. *GROMEX – A scalable and versatile Fast Multipole Method*

for biomolecular simulation. Springer International Publishing, Berlin, Heidelberg, 2020, in press.

[34] J. Kurzak and B. M. Pettitt. Fast multipole methods for particle dynamics. *Molecular simulation*, 32(10-11):775–790, 2006.

[35] C. Kutzner, D. van der Spoel, M. Fechner, E. Lindahl, U. Schmitt, B. de Groot, and H. Grubmüller. Speeding up parallel GROMACS on high-latency networks. *J. Comput. Chem.*, 2007.

[36] C. Kutzner, R. Apostolov, B. Hess, and H. Grubmüller. Scaling of the GROMACS 4.6 molecular dynamics code on SuperMUC. In M. Bader, A. Bode, and H. J. Bungartz, editors, *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, pages 722–730. IOS Press, Amsterdam/Netherlands, 2014.

[37] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller. More bang for your buck: Improved use of GPU nodes for GROMACS 2018. *J. Comput. Chem.*, 40(27):2418–2431, 2019.

[38] T. J. Lane, D. Shukla, K. A. Beauchamp, and V. S. Pande. To milliseconds and beyond: challenges in the simulation of protein folding. *Curr. Opin. Struct. Biol.*, 23(1):58–65, 2013.

[39] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *SC '09: Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009. Association for Computing Machinery.

[40] M. T. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. V. Kal, R. D. Skeel, and K. Schulten. NAMD: a Parallel, Object-Oriented Molecular Dynamics Program. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(4):251–268, 1996.

[41] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53, Mar. 2008.

[42] C. Niedermeier and P. Tavan. A structure adapted multipole method for electrostatic interactions in protein dynamics. *J. Chem. Phys.*, 101(1):734–748, 1994.

[43] S. Páll and B. Hess. A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput. Phys. Commun.*, 2013.

[44] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl. Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In S. Markidis and E. Laure, editors, *Lect. Notes Comput. Sci. 8759, EASC 2014*, pages 1–25. Springer International Publishing Switzerland, 2015.

[45] M. Patra, M. Karttunen, M. Hyvnen, E. Falck, P. Lindqvist, and I. Vattulainen. Molecular dynamics simulations of lipid bilayers: Major artifacts due to truncating electrostatic interactions. *Biophys. J.*, 84(6):3636–3645, 2003.

[46] F. Paul, C. Wehmeyer, E. T. Abualrous, H. Wu, M. D. Crabtree, J. Schöneberg, J. Clarke, C. Freund, T. R. Weikl, and F. Noé. Protein-peptide association kinetics beyond the seconds timescale from atomistic simulations. *Nat. Commun.*, 8(1):1–10, 2017.

[47] D. Potter, J. Stadel, and R. Teyssier. PKDGRAV3: Beyond trillion particle cosmological simulations for the next era of galaxy surveys. *Comput. Astrophys. and Cosmology*, 4(1):2, 2017.

[48] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker. Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald. *J. Chem. Theory Comput.*, 9(9):3878–3888, 2013.

[49] H. Schreiber and O. Steinhauser. Molecular dynamics studies of solvated polypeptides: Why the cut-off scheme does not work. *Chemical Physics*, 168 (1):75–89, 1992.

[50] C. R. Schwantes, R. T. McGibbon, and V. S. Pande. Perspective: Markov models for long-timescale biomolecular dynamics. *J. Chem. Phys.*, 141(9):090901–7, 2014.

[51] D. S. Shamshirgar, R. Yokota, A.-K. Tornberg, and B. Hess. Regularizing the fast multipole method for use in molecular simulation. *J. Chem. Phys.*, 151 (23):234113, 2019.

[52] D. E. Shaw, J. Grossman, J. A. Bank, B. Batson, J. A. Butts, J. C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton, et al. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 41–53. IEEE, 2014.

[53] T. Takahashi, C. Cecka, W. Fong, and E. Darve. Optimizing the multipoletolocal operator in the fast multipole method for graphical processing units. *Int. J. Numer. Methods Eng.*, 89:105 – 133, 01 2012.

[54] R. J. A. Tough and A. J. Stone. Properties of the regular and irregular solid harmonics. *J. Phys. A: Math. Gen.*, 10(8):1261, 1977.

[55] V. Volkov. Better performance at lower occupancy. In *Proceedings of the GPU technology conference, GTC*, volume 10, page 16. San Jose, CA, 2010.

[56] C. A. White and M. Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *J. Chem. Phys.*, 105 (12):5061–5067, 1996.

[57] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[58] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comput. Phys.*, 196(2):591–626, 5 2004.

[59] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, and K. Yasuoka. Fast multipole methods on a cluster of GPUs for the meshless simulation of turbulence. *Comput. Phys. Commun.*, 180(11):2066 – 2078, 2009.

[60] N. Yoshii, Y. Andoh, and S. Okazaki. Fast multipole method for three-dimensional systems with periodic boundary condition in two directions. *J. Comput. Chem.*, 41(9):940–948, 2020.

II.3 A GPU-accelerated Fast Multipole Method for GROMACS: performance and accuracy

In the following work, we presented a very detailed performance and accuracy evaluation of the FMM for biomolecular simulations in GROMACS. I created all FMM benchmarks and generated all resulting plots. The PME benchmarks and figures depicting a few benchmark systems were created by Dr. Carsten Kutzner. I wrote the Introduction and Conclusions sections. The remaining written content of the paper was created by Dr. Carsten Kutzner and by me in an alternating writing process, in which we iteratively enhanced the existing content. The complete work was subsequently edited by Prof. Dr. Gert Lube and by Prof. Dr. Helmut Grubmüller. This paper has been accepted for publication in Journal of Chemical Theory and Computation (JCTC).

A GPU-accelerated Fast Multipole Method for GROMACS: performance and accuracy

Bartosz Kohnke[1], Carsten Kutzner[1], and Helmut Grubmüller[1]

[1] *Max Planck Institute for Biophysical Chemistry, Theoretical and Computational Biophysics, Am Faßberg 11, 37077 Göttingen*

An important and computationally demanding part of molecular dynamics simulations is the calculation of long-range electrostatic interactions. Today, the prevalent method to compute these interactions is particle mesh Ewald (PME). The PME implementation in the GROMACS molecular dynamics package is extremely fast on individual GPU nodes. However, for large scale multi-node parallel simulations, PME becomes the main scaling bottleneck as it requires all-to-all communication between the nodes; as a consequence, the number of exchanged messages scales quadratically with the number of involved nodes in that communication step. To enable efficient and scalable biomolecular simulations on future exascale supercomputers, clearly a method with a better scaling property is required. The fast multipole method (FMM) is such a method. As a first step on the path to exascale, we have implemented a performance-optimized, highly efficient GPU-FMM and integrated it into GROMACS as an alternative to PME. For a fair performance comparison between FMM and PME, we first assessed the accuracies of the methods for various sets of input parameters. With parameters yielding similar accuracy for both methods, we determined the performance of GROMACS with FMM and compared it to PME for exemplary benchmark systems. We found that FMM with multipole order eight yields electrostatic forces that are as accurate as PME with standard parameters. Further, for typical mixed precision simulations settings, FMM does not lead to an increased energy drift with multipole orders of eight or larger. Whereas a $\approx 50,000$ atom simulation system with our FMM reaches only about a third of the performance with PME, for systems with large dimensions and inhomogeneous particle distribution, e.g. aerosol systems with water droplets floating in vacuum, FMM substantially outperforms PME already on a single node.

1 Introduction

The evaluation of mutual interactions in many body systems is a crucial and limiting task in many scientific fields as biomolecular simulations,[?] astronomy[?] or plasma physics.[?] Here, we consider molecular dynamics (MD) simulations, where electrostatic forces

$$\mathbf{f}_i = q_i \sum_{\substack{j=1 \\ j\neq i}}^{N} q_j \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|^3}, \quad i = 1, ..., N \tag{1}$$

acting on N atoms at positions $\mathbf{x}_i$ with partial charges q_i are calculated to determine new positions of the atoms in subsequent discrete time steps. $\|\cdot\|$ denotes the Euclidean norm. A direct calculation of the forces has $\mathcal{O}(N^2)$ complexity, thus only systems of limited size can be computed directly in equitable time. Additionally, a typical MD simulation employs periodic boundary conditions (PBC) to avoid surface artifacts, making the direct calculation unfeasible even for small systems. In contrast to cosmological calculations, which are usually limited by the available memory due to enormous particle numbers,[?] many interesting biomolecular systems consist of $\mathcal{O}(10^5 - 10^6)$ particles. Recently, however, the demand to study increasingly large systems has grown markedly, and systems of $10^8 - 10^9$ particles could become routine soon.[? ? ? ?] Nevertheless, biomolecular systems, independent of their size, require long trajectories where the length of a time step can be no longer than a few femtoseconds for numerical stability reasons. Thus, the time required to finish one simulation step needs to be shortened to a millisecond or less so that long enough trajectories can be produced in reasonable time. To overcome these bottlenecks, the solution of Eq. **??** requires efficient approximation.

The prevalent method for such approximation in the field is particle mesh Ewald (PME).[?] PME uses Ewald summation to split up the calculation into a short range part, for which all interactions up to a cutoff radius r_c are directly evaluated, and a long range part, which is solved in reciprocal space. To take advantage of fast Fourier transforms (FFTs) for the conversions to and from reciprocal space, the charges are interpolated onto a uniform grid using cardinal B-splines. Higher interpolation orders and finer grids yield higher accuracy for the reciprocal part. PME scales with $\mathcal{O}(N \log N)$ and by construction provides a PBC solution, but does not allow for non-periodic calculations.

MD packages like GROMACS[? ? ?] or NAMD[?] have PME implementations that are highly performance-optimized. With GROMACS, typical MD systems reach iteration rates of $\mathcal{O}(1000)$ steps per second currently,[?] hence all forces are computed in less than a millisecond. However, with increasing parallelization, as required for high performance applications, PME runs into a communication bottleneck. Because the FFTs require all-to-all communication, which implies quadratic scaling with the number of processes, PME scaling breaks down at intermediate number of processes.[? ? ?] A further limitation is that the FFT grid becomes memory intensive, particularly if high accuracy is required or for highly inhomogeneous charge distributions.

An alternative way for rapid evaluation of Coulomb forces is the fast multipole method[?] (FMM), which is not impaired by the aforementioned limitations and even scales with $\mathcal{O}(N)$. Therefore, while PME is fast for small to medium sized MD systems at moderate parallelization, FMM will be competitive for large number of particles, large simulation boxes, inhomogeneous charge distributions, and high parallelization.[? ?] Further, FMM can be used for both periodic and open boundaries.

FMM splits the calculation into a *near field*, which is directly evaluated, and into a *far field*. For the far field, groups of sufficiently separated point charges are combined and described as truncated multipole expansions. The grouping is accomplished by recursively subdividing the simulation box into sub-boxes in an octree fashion, i.e. each parent box is subdivided into eight equal child boxes when increasing the tree depth d. This yields 8^d boxes on the lowermost level. For $d = 0$, there is no subdivision. Interactions between particles residing in the same or in directly neighboring boxes at the lowest tree level are calculated directly as in Eq. (**??**), whereas interactions between particles in distant boxes are approximated via far field calculations. FMM can also allow for direct interactions between particles in boxes with a larger distance from each other. The distance is controlled by the well-separateness criterion ws. Larger ws improves the accuracy of the method but it impairs its performance markedly since Eq. (**??**) scales quadratically with respect to the number of particles.[?] In this work we exclusively consider $ws = 1$; hence, only particles of nearest neighbor boxes interact directly.

For the far field interactions, the inverse distance between charged particles with index i and j is approximated as[?]

$$\frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|} \approx \sum_{l=0}^{p} \sum_{m=-l}^{l} \frac{\|\mathbf{x}_i\|^l}{\|\mathbf{x}_j\|^{l+1}} Y_{lm}^*(\theta_i, \phi_i) Y_{lm}(\theta_j, \phi_j), \tag{2}$$

where Y and Y^* are spherical harmonics and their complex-conjugate, respectively. The multipole order p controls the accuracy of the approximation. FMM achieves linear scaling w.r.t. N by performing hierarchical far field operations on multipoles expanded in octree boxes. Computationally, the most demanding part of the far field evaluation is the Multipole-to-Local (M2L) transformation. It requires $\mathcal{O}(p^2)$ dot products with $\mathcal{O}(p^2)$ complexity, yielding an overall complexity of $\mathcal{O}(p^4)$.

The spherical harmonics based FMM (Eq. **??**) was developed by **?**. Following this, other approximations of the inverse distance have been developed, such as the plane wave expansion approach[?] to reduce operational costs of the M2L operator from $\mathcal{O}(p^4)$ or $\mathcal{O}(p^3)$ to $\mathcal{O}(p^2)$ or the black-box FMM,[?] which utilizes Chebyshev interpolation to minimize the far field representation of the multipoles.

One of the first parallel GPU implementations of the spherical harmonics based FMM[?] used $\mathcal{O}(p^3)$ operators and achieved accuracy dependent speedups of 30–70 relative to a serial run on a single CPU core. Recently, the $\mathcal{O}(p^3)$ M2L operator for a single GPU was optimized further.[?] GPUs were used to speed up the kernel independent FMM[? ?] and the black-box FMM.[?] A single-GPU implementation

of the spherical harmonics FMM[?] was also parallelized over a cluster[?] with 32 GPUs where it reached parallel efficiencies of 44 % for 10^6 particles and 66 % for 10^7 particles. Larger, multi-node, multi-GPU parallelization[?] for a 256 million particle system over 256 GPUs followed. ? and ? presented task based parallelization strategies.

The FMM has been successfully used to compute Coulomb or gravitational interactions in a wide range of applications,[? ? ?] whereas its use for biomolecular simulations is still limited with a few exceptions.[? ?] We have therefore developed, implemented and optimized an FMM for MD simulations with GROMACS.

As GROMACS usually runs in *mixed* precision, using double precision only for accumulation order sensitive tasks, consumer GPUs are extremely attractive for the force computation, as they offer a high single precision FLOP rate at a low price, especially compared to CPUs.[?] Therefore, we implemented the complete FMM workflow on the GPU. Whereas rotational M2L operators with complexity $\mathcal{O}(p^3)$ have been proposed,[?] here we consider an $\mathcal{O}(p^4)$ approach for the M2L operator as it is better suited for GPU parallelization.[?]

Our GPU version is based on the ScaFaCoS FMM,[?] which we fully parallelized with CUDA[?] and optimized for GROMACS. Here, we assess the performance of our GPU FMM implementation[?] and evaluate its accuracy in comparison to GROMACS' PME implementation.

2 Benchmark methods

In a first step, we verified that our CUDA FMM implementation yields accurate energies and forces by comparing against known reference solutions for several input systems. Subsequently, we used typical MD systems to compare FMM vs. PME performance in GROMACS 2019.

2.1 Accuracy of FMM results

The forces and energies computed with the FMM deviate from their exact values mainly due to truncation of the multipole expansion at finite order p, which for small p causes the main contribution to the numerical error. Additionally, the errors in the energies vary due to different accumulation orders in the parallelized reductions. To quantify the magnitude of these errors, we compared FMM derived forces, potentials, and energies with reference solutions.

Given a reference solution v_i, $i = 1, \ldots, N$, with N values of the potential at the atomic positions, or the $3N$ individual scalar values x, y, and z force components,

we estimated the approximation error with the cumulative relative L_2 error norm:

$$L_2^{\text{rel}} := \left(\frac{\sum_{i=1}^{N} (v_i - \tilde{v}_i)^2}{\sum_{i=1}^{N} v_i^2} \right)^{\frac{1}{2}}, \tag{3}$$

where $\tilde{v}_i$ are the approximated values.

2.2 Benchmark systems

To assess the correctness and performance of our FMM implementation we created five benchmark systems, which were then used to check different aspects of our implementation. We first verified that the FMM forces and energies for open and periodic boundaries are correct, than we found out the FMM parameters yielding the same accuracy as the existing GROMACS PME implementation. Finally, we compared the performance of both methods at the same accuracy.

GROMACS benchmark systems were set up with GROMACS[?] 2019 using the AMBER03 force field[?] , the TIP3P[?] water model and an integration time step of 4 fs. Note that this force field and water model are just an example – in fact, all force fields and water models supported by GROMACS can be combined with FMM electrostatics.

Infinite ideal crystal

The "ideal crystal" benchmark represents an infinite lattice of alternating positive and negative elementary charges. The charges were arranged as in a NaCl crystal in a $32 \times 32 \times 32$ nm^3 large box containing alternating $+1e$ and $-1e$ charges at 0.5, 1.5, 2.5, ... 30.5, 31.5 nm in each dimension, in total $32^3 = 32,768$ point charges. The shortest distance between nearest charges is exactly 1.0 nm allowing for direct comparison with an analytical solution.

Considering the PBC, such a system of size $2 \times 2 \times 2$ nm^3 would be sufficient to compare against an analytical solution. However, the number of charges was chosen in a way that allows for flexibility during the tests regarding the choice of parameters. For instance, with PME a larger range of real-space cutoffs that can be used, and with FMM various tree depths $d = 1, 2, 3, 4$ can be tested having a significant number of charges even on the lowest levels.

The potential energies at each charge center were calculated analytically with Madelung's constant M.[?] Its value was obtained by summing a specific, three dimensional Epstein zeta function

$$M(s) = \sum_{x,y,z \in \mathbb{Z}}{}' \frac{(-1)^{x+y+z}}{(x^2 + y^2 + z^2)^s} \tag{4}$$

for the case of $s = 1/2$, where $\sum'$ excludes the origin sum to avoid singularity. The sum is absolutely convergent when summing over expanding cubes.[?] For comparison we used the value $M = -1.74756459\ldots$, which is given with 60 digits in Crandall[?] .

Salt water

Our "salt water" benchmark consists of 16,861 water molecules with 46 Na^+ and 46 Cl^- ions in a $\approx 8 \times 8 \times 8$ nm^3 periodic simulation box, yielding 50,675 atoms in total. We used it to compare PME versus FMM errors and to determine which FMM parameters are needed to obtain a desired accuracy. Considering the Coulomb forces, we expect this system to reasonably well approximate the error behavior of typical MD systems of macromolecules embedded in water. However, setups with highly non-uniform charge distributions, as e.g. membrane systems, could differ in their error distribution and magnitude.

An initial trajectory was generated with cutoffs set to 1 nm. PME was used for electrostatic interactions with a grid spacing of 0.135 nm and 4th order B-spline interpolation.[?] Temperature coupling to a heat bath of 300 K was done with the V-rescale algorithm,[?] while pressure was kept at 1 bar using Berendsen coupling.[?]

Salt water droplet

The "salt water droplet," as shown in Fig. **??**, contains the same number of molecules as the periodic salt water system, but in open boundaries. It was built by centering a snapshot of the above system in a larger box of size $14 \times 14 \times 14$ nm^3. Apart from the fixed volume and therefore variable pressure, the simulation parameters are identical to the periodic case. With open boundaries, the system adopted an approximately spherical shape within ≈ 50 ps.

In principle, the box size is only relevant with PBC; technically, however, we used the box to treat the individual single water molecules that did occasionally evaporate from the droplet, as if they were in PBC, simply to keep them from flying too far away. A 130 ns long trajectory of the droplet was simulated, of which snapshots for later analysis were extracted.

The droplet system with open boundaries allows for computation of the reference Coulomb energy and forces by direct summation. It was, therefore, used assess the correctness of the complete FMM implementation apart from the periodic part, which is computed with an additional lattice operator.

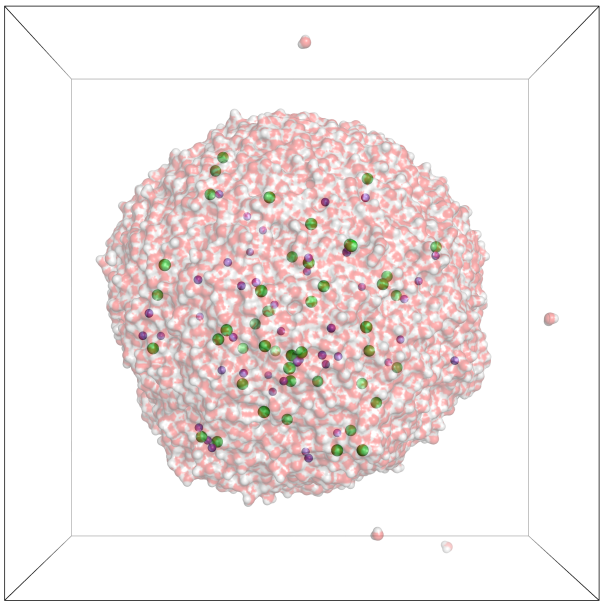


Figure 1: Salt water droplet test system. Water molecules are shown in surface representation (oxygens red, hydrogens white), Na^+ ions in magenta, Cl^- ions in green, simulation box in black.

Aerosol / multi-droplet evaporation system

The "multi-droplet system" (Fig. **??**) was built to demonstrate the advantages of the FMM for systems with highly nonuniform particle distributions, as occur in the atomistic simulation of, e.g., electrospray ionization as a prerequisite for mass spectrometric analysis,[? ? ?] ion mobility spectrometry,[?] laser-induced liquid beam ion desorption,[? ?] and various naturally occurring[?] or artificially produced[?] aerosols.

MD simulations can significantly complement these experiments by providing a detailed picture of the involved processes, e.g. the various aspects of droplet formation and evolution, charge migration, ion/lipid/protein desolvation, collisions with the background gas and gas-phase unfolding. Simulating proteins, lipids, ion, and waters in the gas phase[?] implies spatially extended simulation systems consisting mostly of vacuum.

In the gas phase, due to the lack of shielding, the correct treatment of long-rage electrostatic forces is even more crucial than for fully solvated species to avoid artifacts[?] and to correctly describe experimental conditions.[?] With such extended systems, PME often reaches its limits, as memory requirements become prohibitive for the underlying large FFT grids. Sometimes the use of PME is precluded because for optimal agreement with experiment, open boundaries may be more appropriate than PBC.[?]

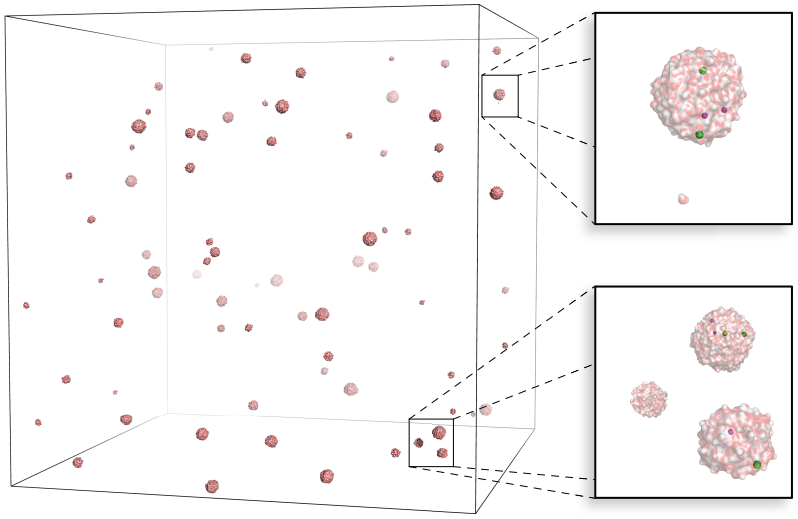


Figure 2: Aerosol / multi-droplet system. Water surface representation as in Fig. **??**; closeups to the right show individual droplets with Na^+ ions in magenta and Cl^- ions in green.

Being a prototype for such sparse systems, our multi-droplet benchmark contains 75 small water droplets in a box of side length 135.6 nm with 108,663 atoms. 63 Na^+ and 63 Cl^- ions were distributed within the droplets. The system was run in the NVE ensemble with PBC. The van der Waals cutoff was set to 2 nm. For PME, to prevent a prohibitively large FFT grid, a Coulomb cutoff of 2.943 nm was used in combination with a grid spacing of 0.353 nm. This results in a Fourier grid of 384^3 points.

Water boxes of different size

To assess how our FMM implementation scales w.r.t. the number of particles N, we have build cubic boxes of edge length 3.13 nm – 67.4 nm containing 1,000 – 10,000,000 TIP3P water molecules,[?] i.e. $N = 3,000$ – $30,000,000$ particles. Benchmarks were run in the NVT ensemble using Berendsen temperature coupling[?] at a reference temperature of 300 K. Coulomb and van der Waals cutoffs were set to 1 nm. With PME, a mesh spacing of 0.135 nm was used with fourth order interpolation.

Random charges

To assess the FMM performance and scaling in a standalone setting, i.e. without being coupled to GROMACS, we used $1000 < N < 286,000,000$ randomly distributed charges in a box of a constant size of 100 nm. FMM standalone tests estimate the overhead introduced by integration of the FMM into GROMACS.

2.3 Benchmarking procedure

All performance benchmarks were run on a node with Intel E5-2630v4 @ 2.2 GHz CPU and NVIDIA RTX 2080Ti GPU running Scientific Linux 7.6 GROMACS 2019 was compiled with GCC 7.4.0, CUDA 10.0, thread-MPI, and AVX2_256 SIMD instructions, and with OpenMP and hwloc[?] 1.11 support. For the runs with PME on the CPU, the FFTW 3.3.7[?] library was used.

For optimum performance in a single-CPU, single-GPU setting,[? ?] we used a single thread-MPI rank with either as many OpenMP threads as physical CPU cores (10), or as many OpenMP threads as CPU hardware threads (20). On modern Intel CPUs, using all available hardware threads can provide a performance benefit of up to $\approx 15\%$ for cases with at least a few thousand atoms per core. We tested both settings in our benchmarks and report the performance of the fastest setting. It turned out that our benchmark systems with 50,000 atoms or more were faster with 20 threads instead of 10.

Additionally, the FMM vs. PME scaling benchmarks were run on a node with 20-core Intel Xeon Gold 6148F CPU and NVIDIA V100-PCIE-32GB GPU running SLES 12.4. Here, GROMACS was compiled with GCC 8.4.0, CUDA 10.1, Intel MPI 2019, and AVX_512 SIMD instructions, and with OpenMP and hwloc 2.1.

Each benchmark ran for several minutes, i.e. several thousand time steps. Because the initial time steps often require long execution times due to memory allocations and load balancing effects, all times were recorded for the second half of each run.

3 Results and Discussion

3.1 FMM convergence and correctness

In this section we quantify the errors resulting from the FMM evaluation of the Coulomb interactions. We will first show that, with increasing order p of the multipole expansion, FMM converges to the correct solution. This was done in two steps. First, we used a system with open boundaries, where the correct solution (within numerical limits) can be obtained by a direct summation. Second, a simple periodic crystal with analytically derived solution was used as a reference to verify the correctness of the FMM PBC solution.

The Coulomb potential V_C for a system of N charges is

$$V_\mathrm{C} = k \cdot \sum_{i}^{N} \sum_{j<i} \frac{q_i q_j}{||\mathbf{r}_i - \mathbf{r}_j||} \tag{5}$$

with $k = 1/(4\pi\epsilon_0)$ and ϵ_0 the vacuum permittivity. Our FMM uses dimensionless

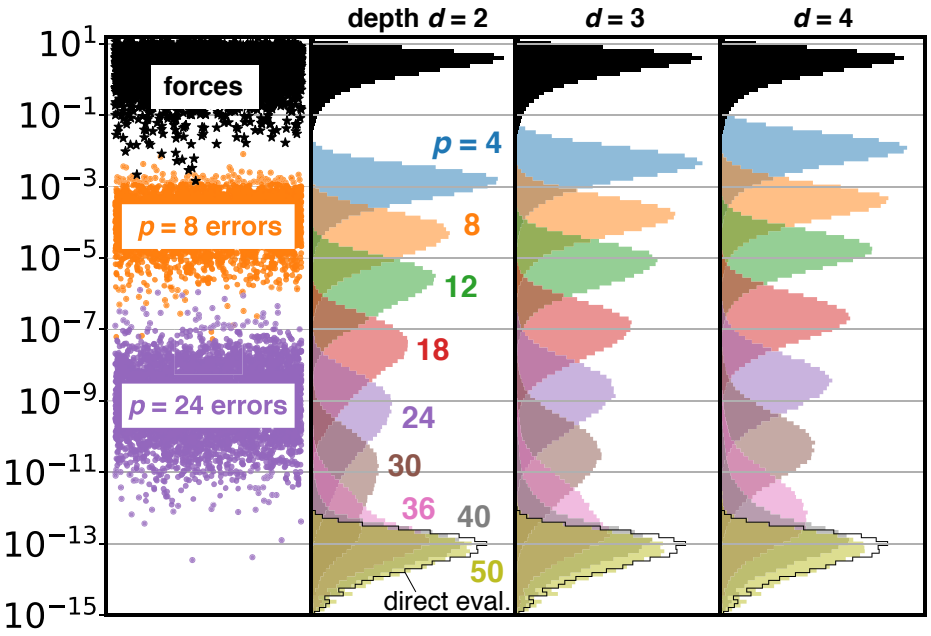


Figure 3: FMM errors for the 50,675 atom salt water droplet (Fig. **??**) using double precision. Left panel shows absolute values of the individual force components (black stars, index on x-axis), and the deviations from the reference values for exemplary cases $p = 8$ (orange dots) and $p = 24$ (purple dots). Colored histograms show the distribution of absolute errors in the forces for multipole approximations $p = 4 - 50$ and tree depths $d = 2$, 3, and 4. For comparison, black histograms show distribution of actual forces (in absolute values). The black outline near the bottom shows the error for directly evaluating all interactions. Note that the black force histograms were scaled by 0.75 to fit in the panels.

values with k set to unity, whereas in GROMACS, $k \approx 138.935\,\mathrm{kJ\,nm/(mol\,}e^2)$, with e the elementary charge. If an axis of a plot shows kJ/mol units for the potential energy and kJ/mol/nm for a force, the GROMACS unit system is used, otherwise energy and forces will be dimensionless.

3.1.1 Comparison to the direct summation for open boundaries

We first asked how accurate the FMM is for open boundaries. For an exemplary snapshot of the salt water droplet (Fig. **??**), we compared the FMM result for different parameters to a reference solution, which was determined by directly summing all Coulomb interactions in double precision.

Figs. **??**–**??** quantify the FMM errors in the Coulomb forces for double and single precision, respectively. The upper rows (black) show the distribution of the $3N$ individual components of the forces f_i^{ref} $(i = 1, \ldots, 3N)$ as absolute values. The colored histograms show error distributions computed from the differences to the

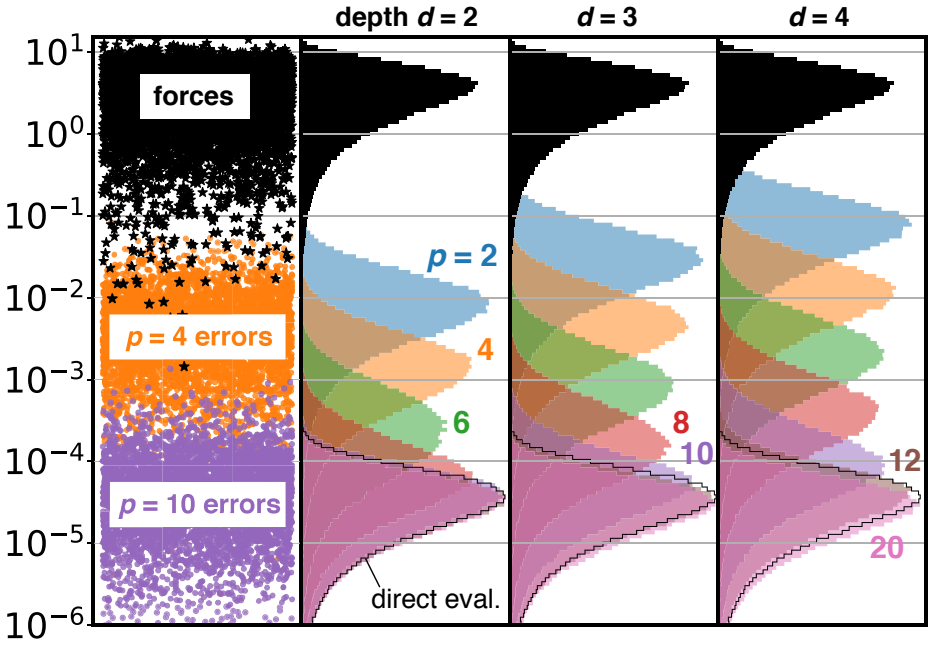


Figure 4: FMM errors for the 50,675 atom salt water droplet (Fig. **??**). Same as Fig. **??**, but for single precision FMM.

reference values $|f_i^{\text{ref}} - f_i^{\text{FMM}}|$ for various multipole orders for depths $d = 2, 3$ and 4.

As can be seen, the force errors decrease exponentially with growing multipole order p and begin to saturate at $p = 40$ and $p = 10$ in double and single precision, respectively. With single precision, as commonly used for MD simulations, increasing the multipole order to $p > 12$ does not result in a further reduction of the error in the Coulomb forces for $d \leq 4$. Increasing the tree depth d by one increases the errors approximately by half an order of magnitude; however, this effect is less pronounced for higher multipole orders.

The black outlined histograms at the bottom of Figs. **??**–**??** quantify the error distributions between different runs of a direct summation. These errors reach maximal relative machine precision,[?] which is 2.22×10^{-16} and 5.96×10^{-8} for double and single precision, respectively. Hence, since the FMM errors with multipole orders $p = 40$ in double and $p = 12$ in single precision saturate in the region of a direct summation error, for both precisions FMM reaches the numerical limits at these multipole orders.

Figs. **??** and **??** show L_2^{rel} error norms of potentials and energies for an exemplary snapshot of the salt water droplet system. In single precision, increasing the multipole order to $p > 12$ does not reduce the error any further as the error reaches the limited machine representation.

In summary, for open boundaries we conclude that FMM forces are as accurate as forces from a direct summation for high multipole orders p. In double precision,

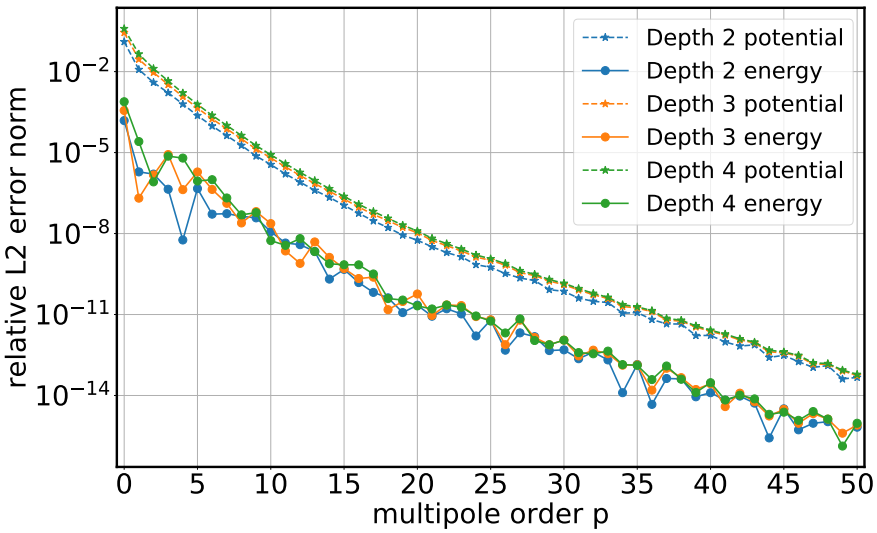


Figure 5: Relative L_2^{rel} error norm (Eq. **??**) of the total electrostatic energy (solid lines with circles) and of the potentials at the atomic positions (dashed lines with stars) for the salt water droplet with open boundaries (double precision).

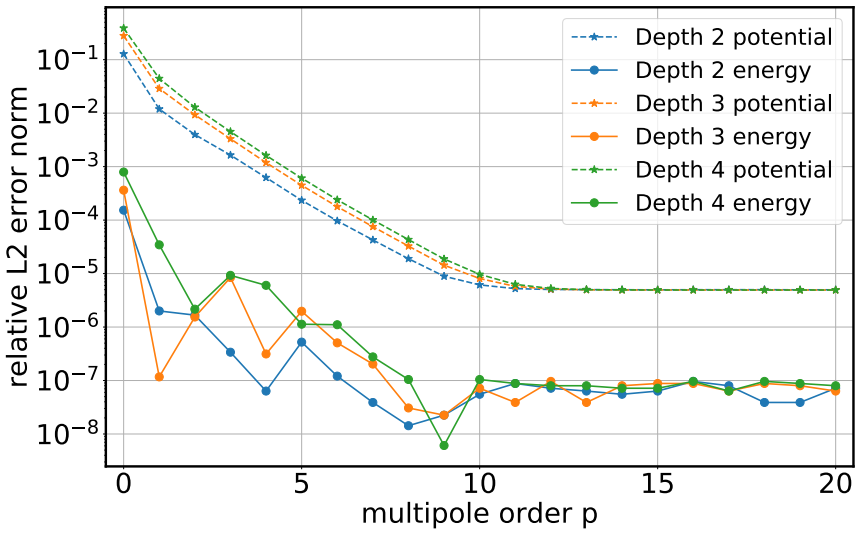


Figure 6: Relative L_2^{rel} error norm (Eq. **??**) of the total electrostatic energy (solid lines with circles) and of the potentials at the atomic positions (dashed lines with stars) for the salt water droplet with open boundaries (single precision).

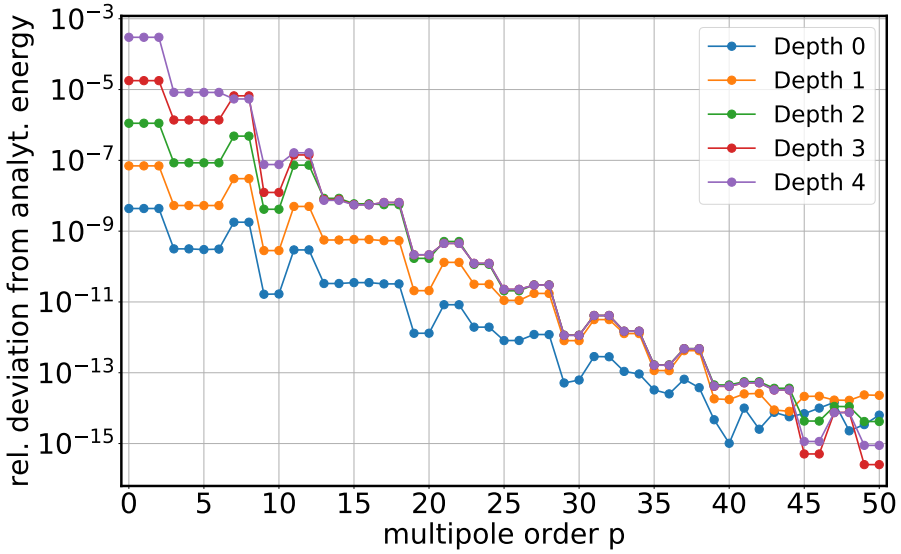


Figure 7: FMM energy error for the ideal crystal (double precision). Circles show the relative deviation of the energy computed with FMM from its correct value as a function of multipole order p and tree depth d.

$p \gtrsim 40$ yields as accurate forces as a direct summation, whereas for single precision, $p \gtrsim 12$ suffices to reach the numerical limits. The relative accuracy of the Coulomb potential energy is about 10^{-7} for $p \geq 8$ in single precision, whereas with double precision, accuracies of 10^{-14} are reached for $p > 40$. For $p < 50$ in double precision and $p < 12$ in single precision, the errors in forces and energies are larger for higher tree depth d.

3.1.2 Comparison to analytic solution for periodic boundaries

Next, we compared the FMM electrostatic energy for the ideal crystal with the analytical results. Fig. **??** shows the relative error in the energy for a double precision computation. The energy error decays exponentially with increasing multipole order. Note that the decay of the energy (compare also Fig. **??** and Fig. **??**) is not strictly monotonic, which follows from the evaluation of the Coulomb integral on cuboids and has been described elsewhere.[? ?] Reaching the relative accuracies at the numerical limit for $p \gtrsim 40$ verifies that the treatment of the periodic boundaries in our FMM implementation is correct and that the FMM approximated energy with full PBC converges to the true value for growing multipole orders.

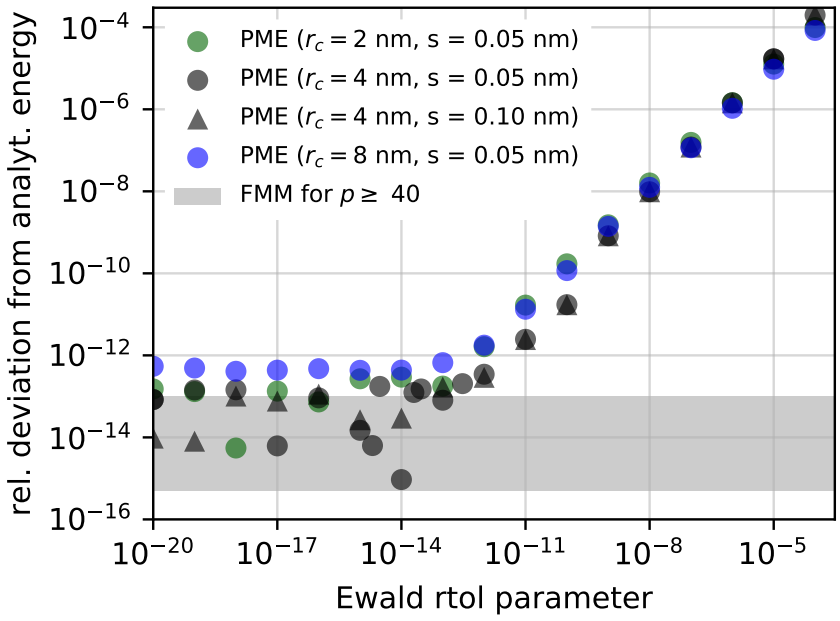


Figure 8: PME energy error for the ideal crystal (double precision). Circles show the relative deviation of the energy computed with PME from its correct value as a function of the `ewald-rtol` parameter for interpolation order 12 for four parameter sets (see legend, r_c = real-space cutoff, s = PME grid spacing) For comparison, the corresponding FMM errors for $p \geq 40$ are indicated by the shaded region (compare Fig. **??**).

3.2 Comparison of FMM to PME

After establishing the correctness of our FMM implementation, we compared it to PME by asking which FMM parameters p and d yield similar accuracy as several representative PME parameter settings, e.g. the spacing s of the Fourier grid and the B-splines interpolation order (also called PME order). In GROMACS' PME implementation, the `ewald-rtol` parameter controls the relative strength of the direct potential at the cutoff r_c and thereby how accurate the real space part is in relation to the reciprocal space part.[?] Smaller values yield a more accurate real space contribution but a less accurate reciprocal space contribution. The default PME parameters use 10^{-5} for `ewald-rtol` which minimizes the error for typical MD settings with cutoffs $r_c \approx 1$ nm and PME grid spacings of $s \approx 0.12$ nm. To reach optimal PME accuracy, however, a much smaller value of the `ewald-rtol` parameter is required in combination with a very fine PME grid and a sufficiently large interpolation order.

3.2.1 PME and FMM for the ideal crystal

Fig. **??** shows how much the energy of the ideal crystal computed using several different PME parameters deviates from the reference values. For the used very fine grids with spacing $s = 0.05 - 0.1$ nm (corresponding to $320^3 - 640^3$ grid points) combined with a high interpolation order of 12, PME accuracy mainly depends on the value of the `ewald-rtol` parameter. For `ewald-rtol` $\lessapprox 10^{-13}$, the energy error achieves roughly 10^{-14}, whereas FMM reaches this error bound for $p \geq 40$. Hence, we have shown that both PME and FMM reach a relative accuracy of $\approx 10^{-14}$ in double precision in a periodic system.

3.2.2 PME vs. FMM for the salt water system

Having shown that FMM and PME yield the same numerical accuracy for the potential energy for a simple periodic system, we switch to a more typical MD setting, namely the periodic salt water box with 50,675 atoms. For this system, we used a reference solution computed with the FMM in double precision using $p = 50$ and $d = 0$.

The colored histograms in Fig. **??** show the errors in the Coulomb forces for various FMM and PME parameters. For PME, we selected four different parameter sets, two of which are representative for typical MD settings, another which pushes the parameters towards maximum accuracy, plus an intermediate one. The "default" set uses the GROMACS default values of PME parameters, which are typical settings for many biomolecular simulations, i.e. a Coulomb cutoff of $r_c = 1.0$ nm with a PME grid spacing of $s = 0.12$ nm and a B-spline interpolation order of 4. The "high precision" set uses $r_c = 1.2$ nm with $s = 0.1$ nm and an interpolation order of 6. We also test a "maximal" parameter set using the largest possible cutoff that still respects the minimum image convention ($r_c = 4.0$ nm) with $s = 0.04$ nm and an interpolation order of 12, which is the highest order supported by GROMACS. The "extreme" parameter set yields a precision between the "high" and "maximal" settings, see legend of Fig. **??**. For each of the four PME parameter sets we have selected the `ewald-rtol` parameter such that it yields the minimal error in the Coulomb energy in double precision, as summarized in Fig. **??**.

For the typical use case with single precision forces, the accuracy of the Coulomb forces for "default" PME parameters is similar to FMM for $p \approx 7$ at $d = 3$. The "high precision" PME parameters require an FMM with $p = 14$.

3.2.3 Periodic boxes with non-cubic geometry

With PBC, our implementation is currently limited to cubic box shapes. Non-cubic simulation boxes require modified octree subdivision,[?] as Eq. **??** converges only if $\|\mathbf{x}_i\| < \|\mathbf{x}_j\|$. As shown in Fig. **??**A, this condition is always fulfilled for cubic

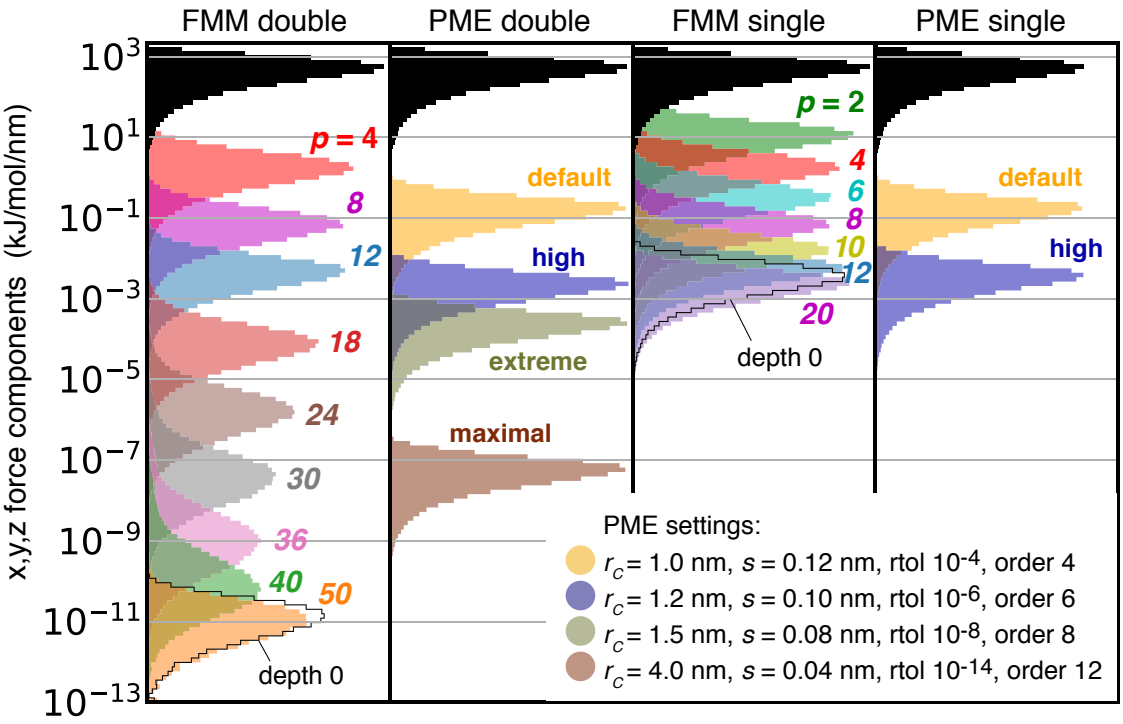


Figure 9: Accuracy of the FMM and PME Coulomb forces for a snapshot of the 50,675 atom periodic salt water system for double precision (left two panels) and single precision (right two panels). Black histograms show distribution of actual forces (in absolute values). For FMM, colored histograms show the distribution of absolute errors in the forces for multipole approximations $p = 2 - 50$ at $d = 3$. For PME, values for four representative parameter sets are shown color-coded (see legend). Note that the black force histograms were multiplied by 0.9 to fit in the panels. The black outline in the FMM panels shows the error for a direct evaluation of all interactions that are in the simulation box ($d = 0$) combined with a $p = 50$ (for double precision, $p = 20$ for single) multipole approximation for the surrounding periodic images.

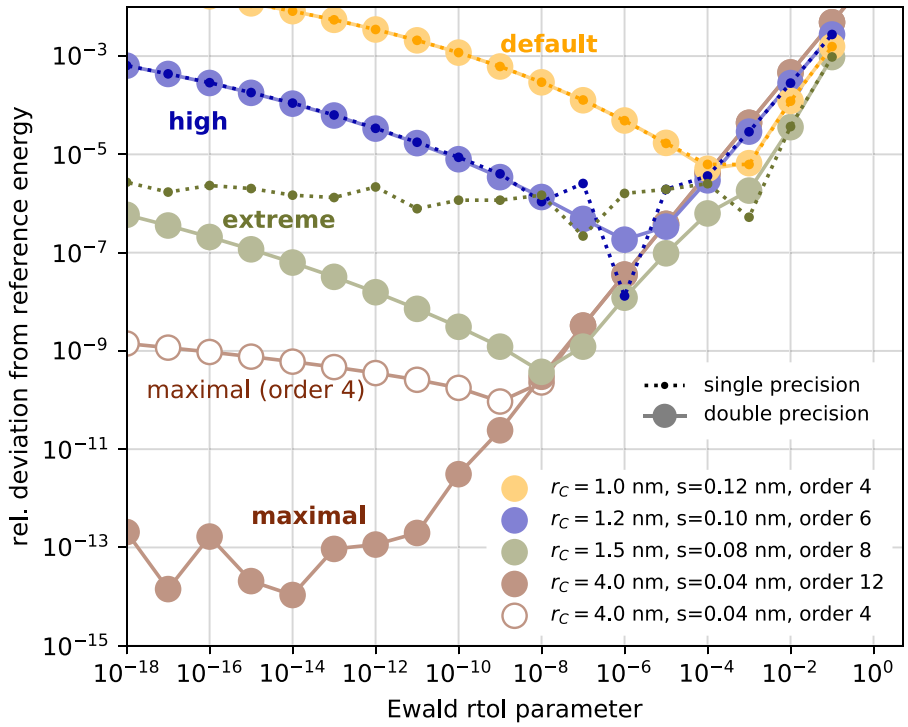


Figure 10: Coulomb energy error for various PME parameters, as in Fig. **??** but for a snapshot of the salt water system for double precision (solid lines with large circles) and single precision (dotted lines with darker small circles). For each combination of r_c, s, and PME order there is one value of the `ewald-rtol` parameter that minimizes the PME error. The reference energy was determined using a double precision FMM calculation with $p = 50$ at $d = 0$. As almost all energy errors are $\geq 10^{-6}$ for single precision, they were omitted from the graph for the "maximal" parameter set (brown).

boxes. Slight deviation from cubicity, Fig. **??**B, do not violate this condition, but a larger ratio $s := \|\mathbf{x}_i\| \,/\, \|\mathbf{x}_j\|$ affects the convergence rate of the approximation. With decreasing s, the approximation error decreases with $\|\mathbf{x}_j - \mathbf{x}_i\|^{-1}\, s^{p+1}$ for given p.[?] As a rough guideline, a rectangular box with a $1.2 : 1$ aspect ratio should achieve a similar accuracy as a cubic box with $p = 8$ (or $p = 12$), if the multipole order is raised to $p = 10$ (or $p = 25$). Slight deviations from cubicity, i.e. a few percent, should however not markedly affect the accuracy at constant p.

3.3 Energy conservation with FMM

For NVE simulations without temperature and pressure control, all employed algorithms must be energy-conserving to prevent a gradual, unphysical heating (or cooling) of the simulation system. But even when a thermostat is in place to absorb excess heat, algorithms should in general not introduce or remove significant

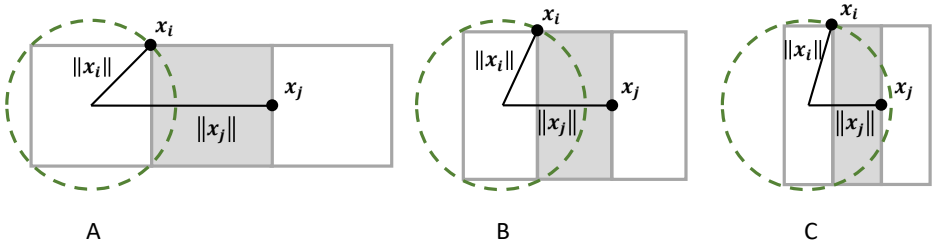


Figure 11: The chosen strict octree subdivision requires the simulation box to be approximately cubic, otherwise the convergence criterion is not fulfilled. **A**: exactly cubic box, **B**: slightly non-cubic box, **C**: extremely non-cubic box. A source particle $\mathbf{x}_i$ and a target particle $\mathbf{x}_j$ are positioned in a way that maximizes the $\|\mathbf{x}_i\| \,/\, \|\mathbf{x}_j\|$ ratio reflecting the worst case scenario.

amounts of heat from the system as that could cause artifacts. In practice, however, slight deviations from perfect energy conservation may be tolerated and in fact many of the employed algorithms contribute (with positive or negative sign) to an overall energy drift. The drift is caused by accumulated numerical and integration errors due to, e.g., the finite integration step size, the finite numerical precision, the constraint algorithm(s), or the various approximations during force calculations.

One such approximation are the pair lists for the Coulomb and van der Waals interactions within the cutoff. For enhanced performance, these lists are constructed from the cutoff plus an added buffer region (called Verlet buffer) so that they do not need to be updated every step. However, with list lifetimes > 1 step, even with such a radial buffer, occasionally a distant nonbonded interaction may be missed, thus contributing to the overall energy drift.[?]

In contrast, for FMM-computed Coulomb interactions, energy drift results from octree space discretization. Whereas PME uses a smooth switching function between interactions computed in direct versus reciprocal space, FMM particles contribute either completely or not at all to an octree box. Hence, particles crossing the octree box boundaries produce small discontinuities in the forces over time.

When substituting PME with FMM we need to make sure that FMM does not increase the total energy drift. Therefore, we have determined the energy drift over time in a typical, mixed precision simulation with PME and compared it to the same simulation with FMM for various FMM parameters. Fig. **??** shows the drift of the total energy for the salt water benchmark with FMM in comparison to PME with "default" parameters ($r_c = 1.0$ nm, $s = 0.12$ nm, fourth order interpolation, `ewald-rtol` $= 10^{-4}$). With FMM, at the depth that yields the highest performance ($d = 3$), the PME default drift level is met for multipole orders $p \geq 8$. The values of the total drift smaller than the black dashed line are due to cancellation of the positive FMM contribution with negative contributions as e.g. result from the water SETTLE constraints.[?] Whereas with double precision and a large enough Verlet buffer, the total drift can be reduced to $< 10^{-7}$ kJ/mol/ps per atom for both PME and FMM (see Figure 11 in **?**), typical mixed precision MD settings yield drifts of

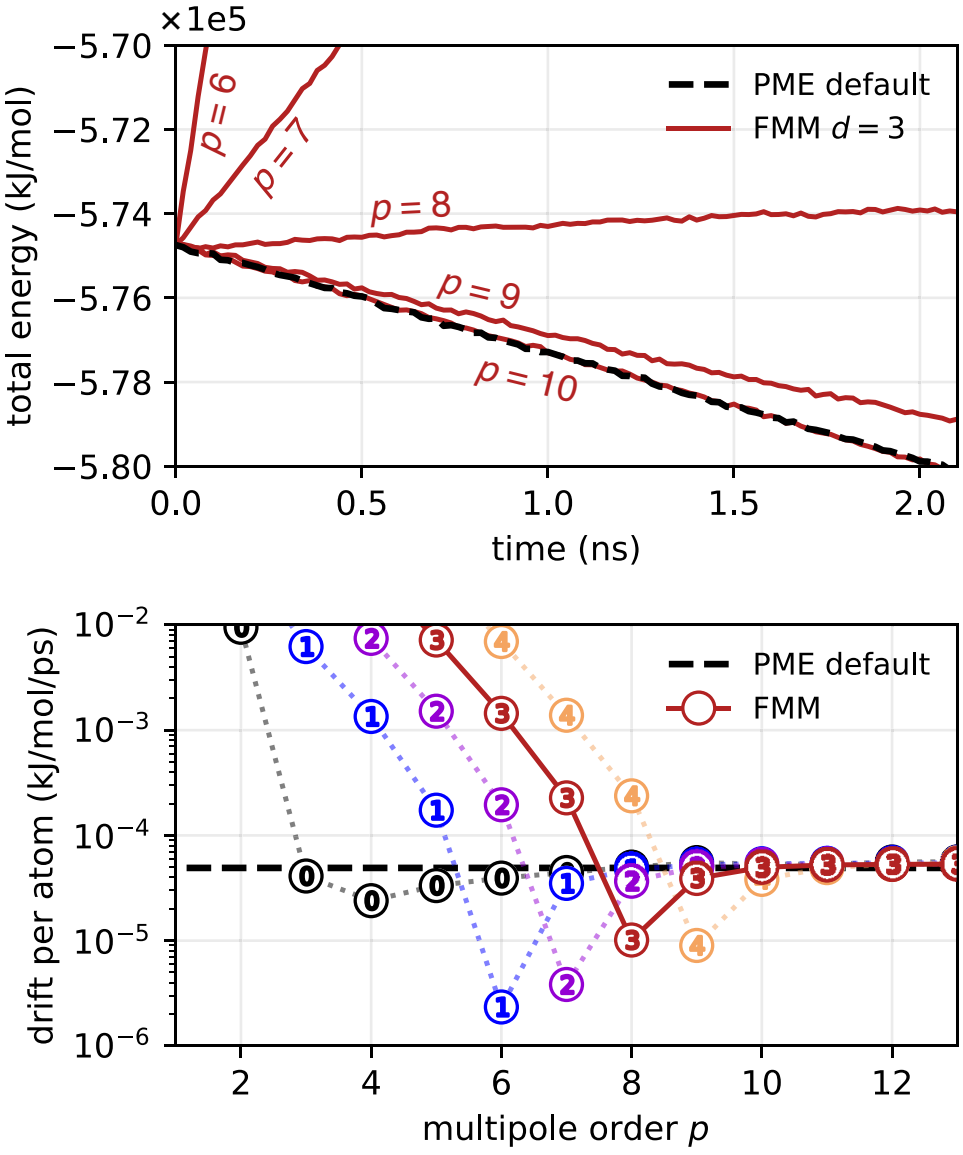


Figure 12: Drift of the total energy at typical mixed precision settings for the periodic salt water system. Dashed black lines show the total (in this case negative) energy drift with PME ($\Delta t = 4$ fs, "default" PME parameters as given in Fig. **??**, default Verlet buffer tolerance of 0.005 kJ/mol/ps). Top panel: Evolution of the total energy with FMM at depth 3 (red) compared to PME (black). Bottom panel: Absolute drift of the total energy derived from a linear fit. At depth $d = 3$ (encircled numbers), which results in optimal FMM performance for this system, for $p \geq 8$, the positive drift component from the FMM does not lead to an increased total drift.

$5 - 8 \times 10^{-5}$ kJ/mol/ps per atom. Regularizing the FMM could help to meet the energy conservation requirements of MD simulation at even lower p, as shown by **?**

3.4 Performance of the GPU FMM in GROMACS

3.4.1 FMM vs. PME performance

With previous tests we have established that the FMM with $p = 8$ and $d = 3$ achieves the same approximation quality as the PME with "default" parameters (see Fig. **??**). Therefore, we compared the performance of the two methods at these parameters.

We first determined the FMM performance as a function of p and d for simulations in mixed precision (Fig. **??**). At $p = 8$, the salt water and multi-droplet benchmark achieves 153 ns/day and 72 ns/day, respectively. For both benchmarks $d = 3$ maximizes the performance. However, the scaling behaviors w.r.t. p notably differ when comparing both systems.

The inhomogeneity of the particles distribution in the multi-droplet system changes the near field to far field calculation intensity ratio. Clustered particles occupy only few FMM boxes, hence, for the far field, the empty boxes are skipped to enhance performance. We can observe that performance dependency on p is significant only for higher multipoles as the calculation is dominated by a very large number of directly interacting particles clustered into only few boxes.

Fig. **??** shows a performance comparison between FMM and PME for both systems. For the periodic salt water system, GROMACS with GPU FMM achieves about a third of the GPU PME performance. The situation reverses for the strongly inhomogeneous multi-droplet system: here, FMM outperforms PME by more than a factor of two.

We finally compared the FMM and PME scaling behavior w.r.t. the number of particles for $N = 3,000$ – $30,000,000$. To ensure optimal scaling, we determined the proper FMM depth for each system size at $p = 8$. As can be seen in Fig. **??**, for both methods we can identify two different slopes with polynomial scaling $\mathcal{O}(N^\alpha)$, where α describes the slope of the curve. For small systems ($N < 30,000$) α is approximately 0.5. This indicates that with growing N in this region the GPU utilization increases leading to a better scaling behavior than linear. For $N > 30,000$ both methods achieve $\alpha \approx 1$ on the Tesla V100 GPU with 32 GB memory in the entire tested particle range. However, when using the RTX 2080Ti GPU with 11 GB memory the scaling begins to worsen already by $\approx 300,000$ particles. Here the FMM scales slightly better ($\alpha = 1.02$) whereas the PME achieves $\alpha = 1.08$ indicating performance decrease due to higher memory requirement.

From the FMM standalone tests, also shown in Fig. **??**, we can clearly see that

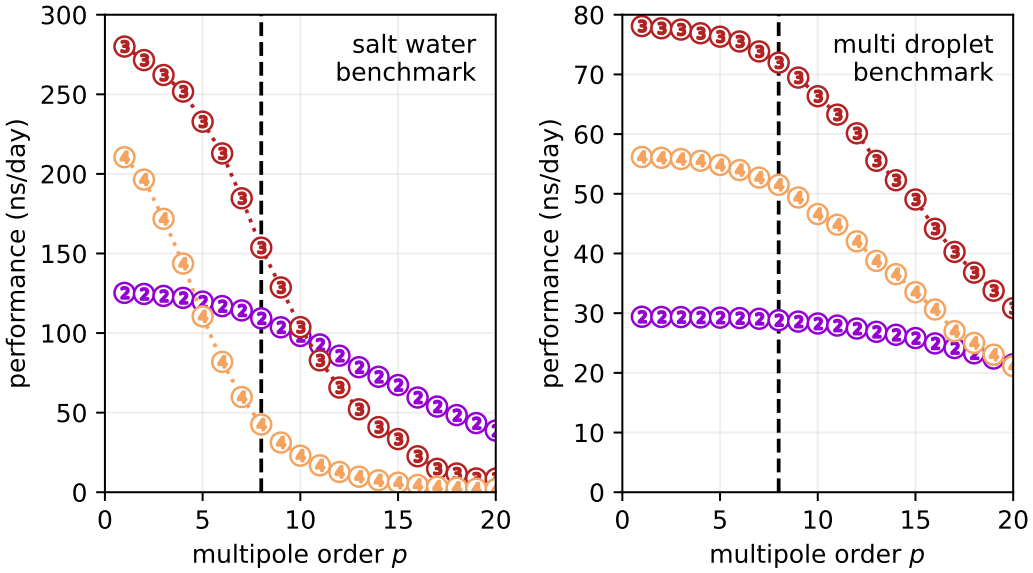


Figure 13: GROMACS performance with FMM electrostatics for the 50,675 atom periodic salt water system (left) and for the 108,663 atom aerosol/multi-droplet benchmark (right). Encircled numbers indicate FMM tree depth. With $p = 8$ as indicated by the dashed vertical line, FMM offers an accuracy of the electrostatic interactions that is comparable to the "default" PME parameter set (i.e. $r_c = 1.0$ nm, PME grid spacing $s = 0.12$ nm, fourth order interpolation, see Fig. **??**). Benchmarks were run with 20 OpenMP threads on the CPU.

our FMM is tightly integrated into the GROMACS time stepping over the whole tested N range, as the runtimes of the FMM with GROMACS are not significantly longer than the FMM standalone runtimes.

Furthermore, on the Tesla V100 GPU, with the standalone FMM implementation we were able to run performance tests for even larger number of particles as, in contrast to PME, where the simulation box size is limited by the available memory, FMM is limited only by the number of particles. Fig. **??** shows that standalone FMM scales linearly up to $\approx 270,000,000$ and $\approx 160,000,000$ particles in single and double precision, respectively. The ability to perform efficient double precision calculations on GPU introduces a new asset as GROMACS is limited to run double precision simulations only on CPU.

3.4.2 Comparison to other FMM implementations

Finally, we compared the performance of our implementation to GemsFMM,[?] which is another GPU FMM implementation written in CUDA. It uses spherical harmonics for the far field evaluation and $\mathcal{O}(p^4)$ operators to shift and transform

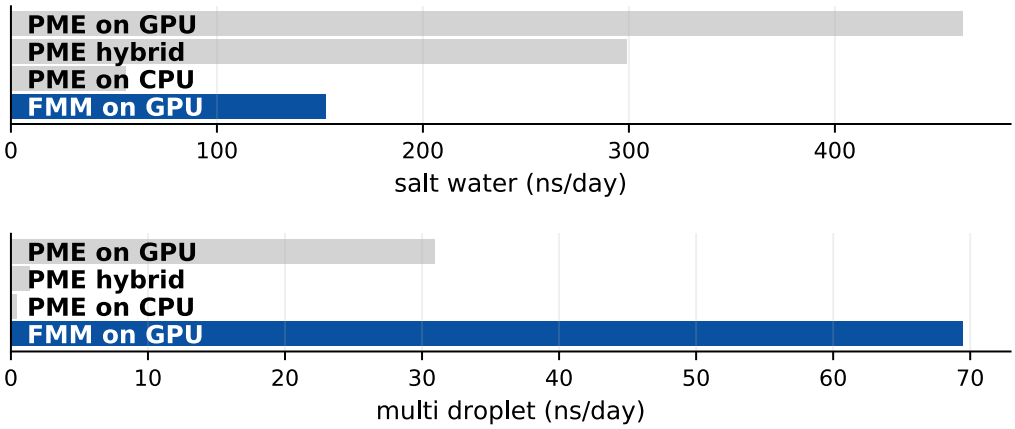


Figure 14: FMM versus PME performance in GROMACS for the salt water (top) and the multi droplet (bottom) benchmark. Settings were chosen such that PME and FMM yield a similar accuracy of the electrostatic forces as well as a comparable energy drift. FMM used $p = 8$ and $d = 3$, whereas PME used the "default" parameter set ($r_c = 1.0$ nm, PME grid spacing $s = 0.12$ nm, fourth order interpolation, see Fig. **??**). For the multi droplet system, for optimal PME performance, both r_c and s were scaled by a factor of 2.943, which leaves PME accuracy essentially unchanged.

the moments. Unfortunately, we were unable to find any other complete GPU-FMM implementations (i.e. that compute both the far field and the near field) that can be tested and provide verifiable results.

Fig. **??** compares FMM runtimes for particle numbers $N = 10^4 - 10^7$. The optimal depth for each N was chosen separately for each implementation to ensure optimal performance. Both implementations show a linear scaling w.r.t. N. The FMM implementation described in this work outperforms GemsFMM by a factor of 5.5 to 13.

4 Conclusions and outlook

Here we have assessed the accuracy and performance of our GPU FMM described in detail in **?** We demonstrated that our implementation provides correct electrostatic energies and forces for single and double numerical precisions by comparing to high-precision reference solutions for open and periodic systems. Using benchmark systems of various sizes and compositions, ranging from 3,000 to 286 M particles, we measured and compared FMM to PME performances in GROMACS on up-to-date GPU models.

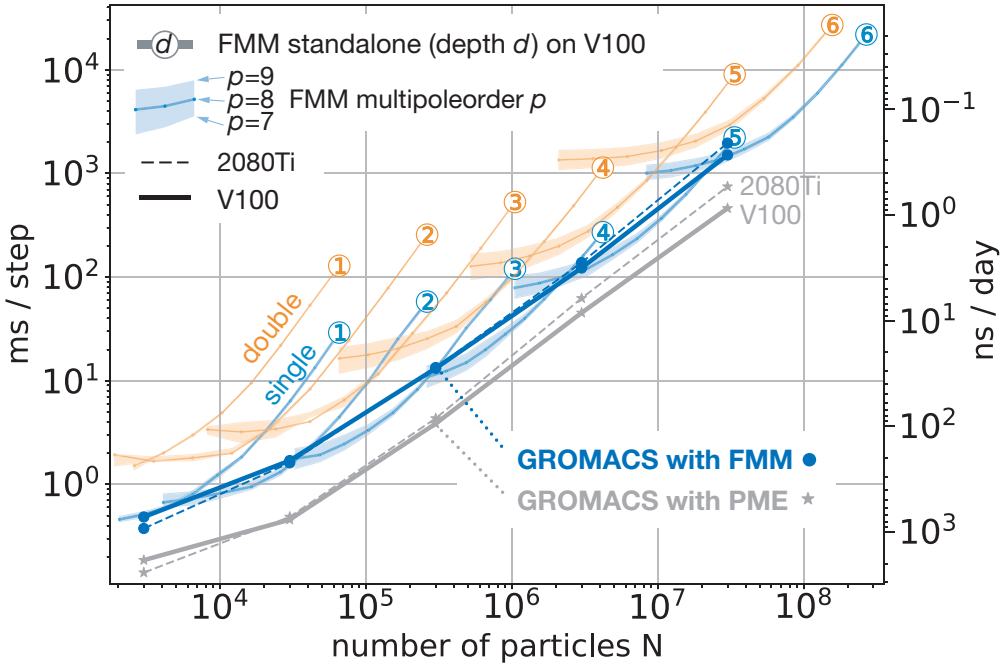


Figure 15: FMM and PME scaling w.r.t. system size N for up to 268 million charges. Benchmarks were run on an NVIDIA Tesla V100 GPU with 32 GB RAM (solid lines) and on an RTX 2080Ti GPU (dashed lines). Blue (single precision) and orange (double precision) colors denote FMM standalone timings for the random charges benchmark (left scale) with depths $d = 1$ – 6 (encircled numbers) and multipole order $p = 8$, whereas the lower and upper boundaries of the shaded regions indicate timings for $p = 7$ and $p = 9$. Grey and dark blue lines show wall clock time per MD step (left scale) and resulting GROMACS performance (right scale) for PME (grey stars) and FMM (blue circles) for waterboxes of different size. GROMACS benchmarks were run on a 10-core E5-2630v4 node with RTX 2080Ti GPU (dashed lines) and on a 20-core Xeon Gold 6148F with V100 GPU (solid lines) with all nonbonded interactions offloaded to the GPU.

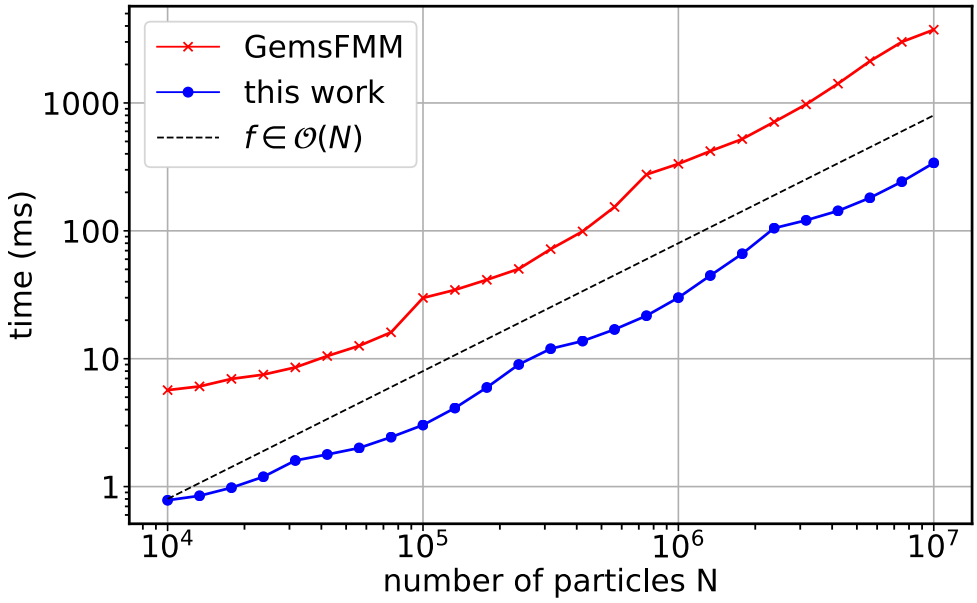


Figure 16: Performance of our FMM (blue) compared to the GemsFMM implementation (red). Shown are the average runtimes for a single complete FMM evaluation (far field plus near field) at $p = 8$ on an NVIDIA RTX 2080 GPU. The black dashed line depicts linear scaling.

As a prerequisite to calculate Coulomb interactions in MD simulations with the FMM, as well as for a proper performance comparison, we have determined the FMM parameters that yield as accurate results as typical PME settings. For a representative biomolecular simulation system of about 50,000 particles in size, a multipole order of seven yields a similar accuracy for the Coulomb energies and forces as standard PME parameters in a mixed precision simulation. The error distribution for the Coulomb forces is comparable for both solvers. Limiting the energy drift to the level present in a standard PME simulation, requires to raise the the multipole order to about eight.

For typical biomolecular systems (proteins in solution) of up to 30 million particles in size, the GROMACS 2019 performance with our CUDA FMM is about a third of that with PME on a single GPU node. However, for systems with larger dimensions and nonuniform particle distributions, as our $\approx 100,000$ atom aerosol/multi-droplet example, FMM easily outperforms PME already at small particle numbers. Here, the huge memory requirements for the FFT grid become the limiting factor for PME.

GemsFmm is a completely independent FMM implementation, which also runs exclusively on the GPU and which uses the same operators for the far field evaluations as our implementation. Our GPU FMM outperforms GemsFMM by a factor of about eight. Unfortunately, further comparisons were not possible because we did not find additional ready-to-use FMM codes that provide verifiable results.

One of the drawbacks of the FMM is that it does not intrinsically allow for non-

cubic simulation boxes with periodic boundaries. For non-cubic boxes the governing octree structure of the FMM would have to be re-designed,[?] requiring further optimizations if the level of achieved performance is to be maintained. Moreover, for typical biomolecular simulation systems of proteins in solution, the single-node GPU FMM is still slower than the highly optimized GROMACS GPU PME implementation; however, single-node GPU FMM can handle larger particle systems and larger simulation boxes.

One of the advantages of FMM electrostatics over PME is that also open boundaries can be handled, however, the FMM's main strength will become apparent on larger exascale clusters of GPU nodes, where PME scaling breaks down due to its inherent communication bottleneck. In combination with the demonstrated high single-GPU performance of this implementation, the performance of a parallelized FMM should eventually beat PME. For large sparse systems, FMM already outperforms PME on a single GPU. Additionally, due to FMM's flexible octree structure that allows to easily evaluate local energy differences, λ–dynamics calculations, as needed for MD simulations at constant pH, can be implemented without much computational overhead.[?]

The next step towards higher FMM performance will be a parallel implementation for multiple GPUs. As the FMM communication requirement is small compared to PME, we expect a parallelized FMM to scale significantly better than PME with the number of GPUs. Additionally, harnessing new CUDA programming features like persistent threads and CUDA Graphs should be beneficial also for single-node GPU performance. Considering large sparse systems, additional optimizations should yield even more speedup because the current implementation was only slightly adopted to handle non-uniform particle distributions.

Appendix

A modified version of GROMACS that includes our CUDA FMM is available for download; please follow the instructions at `https://www.mpibpc.mpg.de/grubmueller/sppexa`.

Acknowledgments

This study was supported by the DFG priority programme *Software for Exascale Computing* (SPP 1648). The multi-droplet system was provided by Frank Wiederschein. Many thanks to Ivo Kabadshow for sharing his insights on FMM error behavior.

References

[] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[] B. Abel, A. Charvat, U. Diederichsen, M. Faubel, B. Girmann, J. Niemeyer, and A. Zeeck. Applications, features, and mechanistic aspects of liquid water beam desorption mass spectrometry. *Int. J. Mass Spectrom.*, 243(2):177–188, 2005.

[] M. J. Abraham and J. E. Gready. Optimization of parameters for molecular dynamics simulation using smooth particle-mesh Ewald in GROMACS 4.5. *J. Comput. Chem.*, 32(9):2031–2040, 2011.

[] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, pages 19–25, 2015. ISSN 2352-7110. doi: 10.1016/j.softx.2015.06.001.

[] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based fmm for heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, 28(9):2608–2629, 2016.

[] Y. Andoh, N. Yoshii, K. Fujimoto, K. Mizutani, H. Kojima, A. Yamada, S. Okazaki, K. Kawaguchi, H. Nagao, K. Iwahashi, F. Mizutani, K. Minami, S.-i. Ichikawa, H. Komatsu, S. Ishizuki, Y. Takeda, and M. Fukushima. Modylas: A highly parallelized general-purpose molecular dynamics simulation program for large-scale systems with long-range forces calculated by fast multipole method (fmm) and highly scalable fine-grained new parallel processing algorithms. *Journal of Chemical Theory and Computation*, 9(7):3201–3209, 2013. doi: 10.1021/ct400203a. URL `https://doi.org/10.1021/ct400203a`. PMID: 26583997.

[] Y. Andoh, N. Yoshii, and S. Okazaki. Extension of the fast multipole method for the rectangular cells with an anisotropic partition tree structure. *J. Comput. Chem.*, 41(14):1353–1367, 2020. doi: 10.1002/jcc.26180. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.26180`.

[] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, and G. Sutmann. Comparison of scalable fast methods for long-range interactions. *Phys. Rev. E*, 88:063308, Dec 2013. doi: 10.1103/PhysRevE.88.063308. URL `https://link.aps.org/doi/10.1103/PhysRevE.88.063308`.

[] H. J. Berendsen, J. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.*, 81(8):3684–3690, 1984.

[] P. Blanchard, B. Bramas, O. Coulaud, E. Darve, L. Dupuy, A. Etcheverry, and G. Sylvand. ScalFMM: A Generic Parallel Fast Multipole Library. In *SIAM Conference on Computational Science and Engineering (SIAM CSE 2015)*, Salt Lake City, United States, 2015. URL `https://hal.inria.fr/hal-01135253`.

[] J. A. Board, C. W. Humphres, C. G. Lambert, W. T. Rankin, and A. Y. Toukmaji. Ewald and multipole methods for periodic n-body problems. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. E. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 459–471. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-642-58360-5.

[] L. Bock, C. Blau, G. Schröder, I. Davydov, N. Fischer, H. Stark, M. Rodnina, A. Vaiana, and H. Grubmüller. Energy barriers and driving forces in tRNA translocation through the ribosome. *Nat. Struct. Mol. Biol.*, 20:1390–1396, 2013. URL `http://hdl.handle.net/11858/00-001M-0000-0014-F59E-4`.

[] D. Borwein, J. M. Borwein, and K. F. Taylor. Convergence of lattice sums and madelungs constant. *J. Math. Phys.*, 26(11):2999–3009, 1985. doi: 10.1063/1.526675. URL `https://doi.org/10.1063/1.526675`.

[] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: A generic framework for managing hardware affinities in hpc applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010, 18th Euromicro International Conference on*, pages 180–186. IEEE, 2010.

[] G. Bussi, T. Zykova-Timan, and M. Parrinello. Isothermal-isobaric molecular dynamics using stochastic velocity rescaling. *J. Chem. Phys.*, 130(7):074101, 2009.

[] C. Caleman, J. S. Hub, P. J. van Maaren, and D. van der Spoel. Atomistic simulation of ion solvation in water explains surface preference of halides. *PNAS*, 108(17):6838–6842, 2011.

[] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *J. Comput. Phys.*, 155(2):468 – 498, 1999. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1999.6355. URL `http://www.sciencedirect.com/science/article/pii/S0021999199963556`.

[] R. E. Crandall. New representations for the madelung constant. *Experimental Mathematics*, 8(4):367–379, 1999. doi: 10.1080/10586458.1999.10504625. URL `https://doi.org/10.1080/10586458.1999.10504625`.

[] C. D. Daub and N. M. Cann. How are completely desolvated ions produced in electrospray ionization: insights from molecular dynamics simulations. *Anal. Chem.*, 83(22):8372–8376, 2011.

[] J. M. Dawson. Particle simulation of plasmas. *Rev. Mod. Phys.*, 55:403–447, Apr 1983.

[] Y. Duan, C. Wu, S. Chowdhury, M. C. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo, T. Lee, J. Caldwell, J. Wang, and P. Kollman. A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations. *J. Comput. Chem.*, 24(16): 1999–2012, 2003.

[] U. Essmann, L. Perera, M. Berkowitz, T. Darden, and H. Lee. A smooth particle mesh Ewald method. *J. Chem. Phys.*, 1995. URL `http://dx.doi.org/10.1063/1.470117`.

[] W. Fong and E. Darve. The black-box fast multipole method. *J. Comput. Phys.*, 228(23):8712 – 8725, 2009. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2009.08.031. URL `http://www.sciencedirect.com/science/article/pii/S0021999109004665`.

[] M. Frigo and S. G. Johnson. The design and implementation of fftw3. *Proc. IEEE*, 93(2):216–231, 2005.

[] A. G. Garcia, A. Beckmann, and I. Kabadshow. *Accelerating an FMM-Based Coulomb Solver with GPUs*, pages 485–504. Springer International Publishing, Berlin, Heidelberg, 2016. ISBN 978-3-319-40528-5. doi: 10.1007/978-3-319-40528-5_22.

[] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325 – 348, 1987. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(87)90140-9. URL `http://www.sciencedirect.com/science/article/pii/0021999187901409`.

[] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the laplace equation in three dimensions. *Acta Numerica*, 6:229269, 1997. doi: 10.1017/S0962492900002725.

[] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *J. Comput. Phys.*, 227(18):8290 – 8313, 2008. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2008.05.023. URL `http://www.sciencedirect.com/science/article/pii/S0021999108002921`.

[] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *J. Chem. Theory Comput.*, 2008.

[] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.*, 79(2):926–935, 1983.

[] J. Jung, W. Nishima, M. Daniels, G. Bascom, C. Kobayashi, A. Adedoyin, M. Wall, A. Lappala, D. Phillips, W. Fischer, C.-S. Tung, T. Schlick, Y. Sugita, and K. Y. Sanbonmatsu. Scaling molecular dynamics beyond 100,000 processor cores for large-scale biophysical simulations. *J. Comput. Chem.*, 2019.

[] I. Kabadshow. *Periodic boundary conditions and the error-controlled fast multipole method*, volume 11. Forschungszentrum Jülich, 2012.

[] D. Kim, N. Wagner, K. Wooding, D. E. Clemmer, and D. H. Russell. Ions from solution to the gas phase: a molecular dynamics simulation of the structural evolution of substance P during desolvation of charged nanodroplets generated by electrospray ionization. *J. Am. Chem. Soc.*, 139(8):2981–2988, 2017.

[] B. Kohnke, C. Kutzner, A. Beckmann, R. T. Ullmann, G. Lube, I. Kabadshow, H. Dachsel, and H. Grubmüller. A CUDA Fast Multipole Method with highly efficient M2L far field evaluation. *submitted*, volume(number):pages, 2020.

[] B. Kohnke, T. R. Ullmann, A. Beckmann, I. Kabadshow, D. Haensel, L. Morgenstern, P. Dobrev, G. Groenhof, C. Kutzner, B. Hess, H. Dachsel, and H. Grubmüller. *GROMEX A scalable and versatile Fast Multipole Method for biomolecular simulation*, pages ?–? Springer International Publishing, Berlin, Heidelberg, 2020.

[] L. Konermann, H. Metwally, R. G. McAllister, and V. Popa. How to run molecular dynamics simulations on electrospray droplets and gas phase proteins: Basic guidelines and selected applications. *Methods*, 144:104–112, 2018.

[] K. N. Kudin and G. E. Scuseria. A fast multipole method for periodic systems with arbitrary unit cell geometries. *Chem. Phys. Lett.*, 283(1–2):61–68, 1998.

[] C. Kutzner, D. van der Spoel, M. Fechner, E. Lindahl, U. Schmitt, B. de Groot, and H. Grubmüller. Speeding up parallel GROMACS on high-latency networks. *J. Comput. Chem.*, 2007.

[] C. Kutzner, R. Apostolov, B. Hess, and H. Grubmüller. Scaling of the GROMACS 4.6 molecular dynamics code on SuperMUC. In M. Bader, A. Bode, and H. J. Bungartz, editors, *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, pages 722–730. IOS Press, Amsterdam/Netherlands, 2014. doi: 10.3233/978-1-61499-381-0-722.

[] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller. More bang for your buck: Improved use of GPU nodes for GROMACS 2018. *J. Comput. Chem.*, 40(27):2418–2431, 2019. doi: 10.1002/jcc.26011.

[] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *SC '09: Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587448. doi: 10.1145/1654059.1654118. URL `https://doi.org/10.1145/1654059.1654118`.

[] W. Luedtke, U. Landman, Y.-H. Chiu, D. Levandier, R. Dressler, S. Sok, and M. S. Gordon. Nanojets, electrospray, and ion field evaporation: Molecular dynamics simulations and laboratory experiments. *J. Phys. Chem. A*, 112(40): 9628–9649, 2008.

[] E. G. Marklund and J. L. Benesch. Weighing-up protein dynamics: the combination of native mass spectrometry and molecular dynamics simulations. *Curr. Opin. Struct. Biol.*, 54:50–58, 2019.

[] T. Meyer, V. Gabelica, H. Grubmüller, and M. Orozco. Proteins in the gas phase. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.*, 3(4):408–425, 2013.

[] M. E. Mura and N. C. Handy. Cuboidal basis functions. *Theor. Chim. Acta*, 90(2-3):145–165, 1995.

[] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, Mar. 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL `http://doi.acm.org/10.1145/1365490.1365500`.

[] Y. Ohno, R. Yokota, H. Koyama, G. Morimoto, A. Hasegawa, G. Masumoto, N. Okimoto, Y. Hirano, H. Ibeid, T. Narumi, and M. Taiji. Petascale molecular dynamics simulation using the fast multipole method on k computer. *Computer Physics Communications*, 185(10):2575 – 2585, 2014. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2014.06.004. URL `http://www.sciencedirect.com/science/article/pii/S0010465514002082`.

[] S. Páll and B. Hess. A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput. Phys. Commun.*, 184(12):2641–2650, dec 2013. ISSN 00104655. doi: 10.1016/j.cpc.2013.06.003.

[] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl. Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In S. Markidis and E. Laure, editors, *Lect. Notes Comput. Sci. 8759, EASC 2014*, pages 1–25. Springer International Publishing Switzerland, 2015.

[] J. R. Perilla, B. C. Goh, C. K. Cassidy, B. Liu, R. C. Bernardi, T. Rudack, H. Yu, Z. Wu, and K. Schulten. Molecular dynamics simulations of large macromolecular complexes. *Curr. Opin. Struct. Biol.*, 31:64–74, 2015.

[] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD. *J. Comput. Chem.*, 26(16):1781–1802, 2005. URL `http://www.ks.uiuc.edu/Research/namd/`.

[] D. Potter, J. Stadel, and R. Teyssier. Pkdgrav3: Beyond trillion particle cosmological simulations for the next era of galaxy surveys. *Comput. Astrophys. and Cosmology*, 4(1):2, 2017.

[] T. A. Schoolcraft, G. S. Constable, L. V. Zhigilei, and B. J. Garrison. Molecular dynamics simulation of the laser disintegration of aerosol particles. *Anal. Chem.*, 72(21):5143–5150, 2000.

[] H. Schreiber and O. Steinhauser. Cutoff size does strongly influence molecular dynamics results on solvated polypeptides. *Biochemistry*, 31(25):5856–5860, 1992.

[] D. Shamshirgar, R. Yokota, A.-K. Tornberg, and B. Hess. Regularizing the fast multipole method for use in molecular simulation. *J. Chem. Phys.*, 151(23): 234113, 2019. doi: 10.1063/1.5122859.

[] T. Takahashi, C. Cecka, W. Fong, and E. Darve. Optimizing the multipoletolocal operator in the fast multipole method for graphical processing units. *Int. J. Numer. Methods Eng.*, 89:105 – 133, 01 2012. doi: 10.1002/nme.3240.

[] D. van der Spoel, E. G. Marklund, D. S. Larsson, and C. Caleman. Proteins, lipids, and water in the gas phase. *Macromolecular Bioscience*, 11(1):50–59, 2011.

[] R. Yokota and L. A. Barba. Chapter 9 - treecode and fast multipole method for n-body simulation with cuda. In W. W. Hwu, editor, *GPU Computing Gems Emerald Edition*, Applications of GPU Computing Series, pages 113–132. Morgan Kaufmann, Boston, 2011.

[] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, and K. Yasuoka. Fast multipole methods on a cluster of gpus for the meshless simulation of turbulence. *Comput. Phys. Commun.*, 180(11):2066 – 2078, 2009. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2009.06.009. URL `http://www.sciencedirect.com/science/article/pii/S0010465509001891`.

[] M. Zink and H. Grubmüller. Mechanical properties of the icosahedral shell of southern bean mosaic virus: A molecular dynamics study. *Biophys. J.*, 96:1350–1363, 2009. URL `http://hdl.handle.net/11858/00-001M-0000-000F-9BDD-1`.

III Conclusions and outlook

Within this work, I implemented a complete CUDA-GPU version of the FMM algorithm and evaluated its accuracy and performance. Additionally, I optimized the implementation towards the specific requirements of biomolecular simulations; however, the standalone variant of the code can be used for any approximation of the n-body problem with r^{-1} potential. The implementation utilizes a modern C++11 framework to enable flexible parallelization regarding future multi-node, multi-GPU implementations. Last but not least, I extended the method to support chemical variability via λ-dynamics.

III.1 Accuracy of the CUDA FMM

To facilitate usage of the FMM in MD simulations with desired accuracy, we had to determine proper FMM parameters. To this aim, we verified the correctness of our FMM for different computational scenarios. Starting with open-boundary tests, we compared the FMM solution with a reference solution, which was computed directly in double precision. Afterwards, we used an analytical reference solution for a simple salt crystal to verify that the periodic part of the method yields a correct result, too. From that, we observed that FMM with multipole order $p = 50$ yields an approximation error in the range of the machine precision error. This observation allowed to create a periodic reference solution, which was used to compare the error distributions of the Coulomb forces for PME and FMM with different parameters. From this comparison, we concluded that a multipole order of seven is sufficient to achieve an error similar to PME with default settings. To limit the energy drift to the level of a standard PME simulation, the multipole order of eight is required. This knowledge allowed for a further performance comparison of the FMM and PME for MD simulations.

III.2 Performance of the CUDA FMM

The FMM has been completely ported to GPU and incorporated efficiently into GROMACS to allow using of the method as PME replacement to calculate pairwise Coulombic interactions. The performance of the single-GPU FMM implementation, tested in GROMACS 2019, achieves about a third of highly optimized CUDA PME performance when simulating systems with uniform particle distributions. However, the usage of the FMM is expected to be beneficial when aiming a massive parallelization. As the number of particles in MD simulations is mostly of order $\mathcal{O}(10^4) - \mathcal{O}(10^7)$, a limited number of particles per compute node is expected by a high number of computational nodes. This requires a strong scaling property of the underlying method. Hence, the wall clock times per iteration should be in a latency range of the network communication. In such scenarios, PME reaches its scaling limit as it requires all-to-all communication. FMM does not suffer from

this bottleneck and the underlying GPU implementation should enable an efficient heterogeneous parallelization on future exascale supercomputers.

An efficient usage of the FMM on a computational node in a parallel environment requires considerable flexibility of the underlying implementation as, depending on the hardware specification of a computing node, different portions of the FMM calculation are required to be scheduled as a GPU task. For this reason, I tested several different parallelization approaches with different amount of parallel granularity and different work load distributions. The most performance limiting far field operator, i.e. the M2L operator that has a complexity of $\mathcal{O}(p^4)$, exposes a lot of operator parallelism, however, its efficient GPU parallelization is impeded by its highly non-uniform distributed data. To find an optimal parallelization scheme for efficient utilization of heterogeneous memory layers provided by a GPU, I tested a great number of different approaches. Three of them are described in detail in section II.2. The symmetrical implementation of the M2L operator scales nearly optimal in the entire tested range of p ($1 < p < 21$) and its advantage for a single GPU implementation is evident. However, its future usage in heterogeneous parallel environments is not straightforward as the implementation incorporates a non-local tree parallelism. The dynamic approach exposes a lot single operator parallelism and strictly separates the tree and operator parallelism. Its main bottleneck, launching latencies of the dynamically spawned kernels, can directly be reduced by statically grouping the required interactions. Utilizing such grouping makes this approach suitable for further utilization on heterogeneous clusters. It has also been utilized to enhance the simulations with non-uniform particle distributions as it allows to efficiently skip the non interacting parts in a system.

In addition to the far-field operators, the near-field $\mathcal{O}(n^2)$ part has been also parallelized in a very flexible way allowing for efficient utilization in case of massive parallelization. Similar to the M2L method, the most optimized version combines the tree and the operator parallelism. The compute bound nature of the direct calculation can be clearly seen in this approach as the achieved FLOP rates are in a range of the peak performance of the underlying GPU when a single-node implementation is concerned. However, a different more flexible approach, which exposes more locality, is supposed to be beneficial in large parallel computations. Further, its flexibility also allows for calculations between sparsely populated particle groups in λ-dynamics.

Apart from the mentioned flexible implementation, FMM provides a few intrinsic assets when compared to the PME method. In contrast to the PME method, which is periodic by construction, FMM can handle non-periodic systems with negligible overhead. When considering large dimensional or non-uniformly distributed systems, FMM outperforms PME already at small particle numbers on a single GPU node. For such calculations, the performance of the PME is limited by its large memory requirement for the FFT grid.

The entire FMM implementation can also be used as an efficient standalone library for solving an arbitrary n-body problem, where interactions are described by the r^{-1}

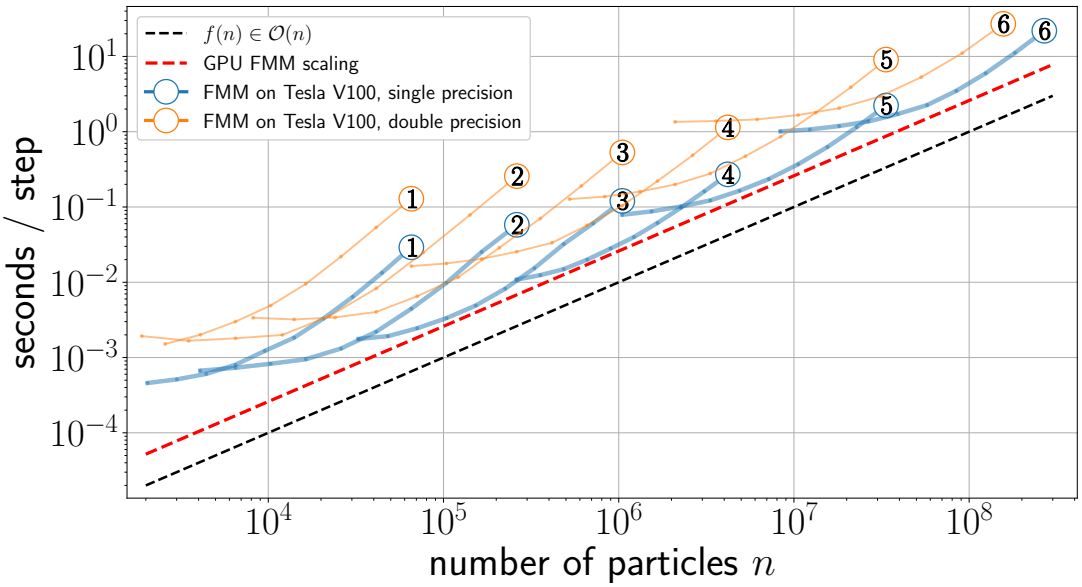


Figure 1: Scaling of the GPU parallelized FMM. The black dashed line shows a reference $\mathcal{O}(n)$ scaling. The red dashed line depicts the approximation of the GPU-FMM scaling in single precision. Blue and yellow lines denote exact FMM timings for one calculation step for different octree depths (encircled numbers) in single and double precision, respectively, for multipole order $p = 8$.

potential. Figure 1 shows the scaling of the GPU parallelized FMM for multipole order $p = 8$. The single-GPU implementation scales linearly with respect to the number of particles in the whole possible particle range, which is only restricted by the available memory. The method is efficiently implemented in a single and double precision what allows for calculations with any desired accuracy up to numerical precision. It also does not rely on any external libraries providing an uncomplicated usage and installation process.

III.3 Parallelization and performance of the λ-FMM

To allow for an efficient computation of additional potentials required by the λ-dynamics algorithm, I incorporated a full λ-dynamics functionality into the existing CUDA GPU implementation of the FMM. To this end, I created a basic GPU version of the method that can treat an arbitrary number of titratable sites with an arbitrary number of states each. The implementation does not impose any restrictions considering the spatial distribution of the atoms belonging to particular sites. The method is not fully optimized yet, but first results show the strength of the FMM in a λ-dynamics environment. For a relatively large system with half a million particles and more than a hundred titratable sites, a straightforward approach to compute all alternative energies requires the repetition of a whole simulation for each state of all sites. This overhead is linear and it is directly proportional to the number of all alternative states in all sites. The λ-FMM computes all required energy terms with

only a factor of 1.5 slower than a single simulation step. Further improvements of the GPU implementation are expected to yield even more speedup compared to the actual implementation.

III.4 Outlook

Further improvements of the single GPU implementation are possible. The most efficient, symmetrical implementation of the M2L operator can still be enhanced by exploiting even more symmetry when aiming the best possible performance. With a new concept of the GPU persistent threads, which provide more synchronization between different tasks, additional latencies can be hidden, especially when considering not efficiently parallelizable stages of the FMM. The treatment of systems with nonuniformly distributed particles can also be enhanced as the existing operators were only slightly adapted to efficiently deal with sparsely populated FMM boxes. To expand the FMM usage scenarios, the method should also be extended to support non-cubic boxes in periodic systems. Following this extension, fractal FMM tree depths should be implemented to minimize the influence of the local quadratic scaling of the method.

When considering future multi-node, multi-GPU parallelization of the FMM, new challenges arise. Whereas FMM will definitely allow to perform efficient multi-node simulations for much larger MD systems (10^8 particles and beyond) as FMM octree structure enables very efficient parallelization, maintaining the strong scaling behavior for limited system sizes, which are even more important, is very challenging. As the number of transistors is doubling nearly every two years and the number of nodes in computing clusters is also growing, present parallel implementations require redesigning of the inter-processes communication. In such scenarios, the communication and memory latencies become the computational bottleneck whereas FLOPS can be considered as negligible. For the next step towards exascaling, a multi-GPU FMM is needed. Such implementation will possibly reflect potential communication bottlenecks in a multi-node environment as the multi-GPU memory hierarchy usage might indicate communication bottlenecks in further parallelizations.

To enable even more efficient treatment of MD systems with titratable sites, the existing λ-dynamics implementation needs to be optimized. Despite the actual, fully working, parallel λ-dynamics version with only a few optimizations serves as a proof of concept, it already achieves a decent performance in calculating alternative energies which are needed for updating λ values. However, the existing implementation requires a complete redesign of memory management since additional sites introduce very sparse and non-uniform memory patterns, which change rapidly depending on the dynamics of the system. Additionally, most far field operators have to be started conditionally. This introduces a large overhead especially in latency bound regime. Nevertheless, we anticipate that further improvements of the parallel implementation of the λ-dynamics will allow for calculations for systems with a very large number of titratable sites with only negligible additional costs.

References

[1] A. W. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(1):85–103, 1985.

[2] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, and G. Sutmann. Comparison of scalable fast methods for long-range interactions. *Phys. Rev. E*, 88:063308, 2013.

[3] J. Barnes and P. Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[4] C. L. Berman. Grid-multipole calculations. *SIAM Journal on Scientific Computing*, 16(5):1082–1091, 1995.

[5] P. Blanchard, B. Bramas, O. Coulaud, E. Darve, L. Dupuy, A. Etcheverry, and G. Sylvand. ScalFMM: A Generic Parallel Fast Multipole Library. In *SIAM Conference on Computational Science and Engineering (SIAM CSE 2015)*, Salt Lake City, United States, 2015.

[6] J. A. Board, C. W. Humphres, C. G. Lambert, W. T. Rankin, and A. Y. Toukmaji. Ewald and multipole methods for periodic n-body problems. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. E. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 459–471. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[7] L. Bock, C. Blau, G. Schröder, I. Davydov, N. Fischer, H. Stark, M. Rodnina, A. Vaiana, and H. Grubmüller. Energy barriers and driving forces in tRNA translocation through the ribosome. *Nat. Struct. Mol. Biol.*, 20:1390–1396, 2013.

[8] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 43–43. IEEE, 2006.

[9] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[10] C. L. Brooks III, B. M. Pettitt, and M. Karplus. Structural and energetic effects of truncating long ranged interactions in ionic and polar fluids. *The Journal of chemical physics*, 83(11):5897–5908, 1985.

[11] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of computational physics*, 155(2):468–498, 1999.

[12] J. M. Dawson. Particle simulation of plasmas. *Rev. Mod. Phys.*, 55:403–447, Apr 1983.

[13] S. Donnini, F. Tegeler, G. Groenhof, and H. Grubmüller. Constant ph molecular dynamics in explicit solvent with λ-dynamics. *Journal of Chemical Theory and Computation*, 7(6):1962–1978, Jun 2011.

[14] M. Eichinger, H. Grubmüller, H. Heller, and P. Tavan. Famusamm: An algorithm for rapid evaluation of electrostatic interactions in molecular dynamics simulations. *Journal of Computational Chemistry*, 18(14):1729–1749, 1997.

[15] W. D. Elliott and J. A. Board, Jr. Fast fourier transform accelerated fast multipole algorithm. *SIAM Journal on Scientific Computing*, 17(2):398–415, 1996.

[16] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen. A smooth particle mesh ewald method. *The Journal of chemical physics*, 103(19):8577–8593, 1995.

[17] P. P. Ewald. Die berechnung optischer und elektrostatischer gitterpotentiale. *Annalen der physik*, 369(3):253–287, 1921.

[18] R. Feynman, R. Leighton, and M. Sands. *The Feynman Lectures on Physics: Mainly mechanics, radiation, and heat.* Number Bd. 1-3 in Addison-Wesley world student series. Addison-Wesley Publishing Company, 1963.

[19] W. Fong and E. Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[20] A. G. Garcia, A. Beckmann, and I. Kabadshow. Accelerating an fmm-based coulomb solver with gpus. In *Software for Exascale Computing-SPPEXA 2013-2015*, pages 485–504. Springer, 2016.

[21] M. González. Force fields and molecular dynamics simulations. *École thématique de la Société Française de la Neutronique*, 12:169–200, 2011.

[22] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[23] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227(18):8290–8313, 2008.

[24] Z. Guo, C. Brooks, and X. Kong. Efficient and flexible algorithm for free energy calculations using the λ-dynamics approach. *J. Phys. Chem. B*, 102(11):2032–2036, 1998.

[25] T. Hansson, C. Oostenbrink, and W. van Gunsteren. Molecular dynamics simulations. *Curr. Opin. Struct. Biol.*, 12(2):190–196, 2002.

[26] P. Henri. *Sur le problème des trois corps et les équations de la dynamique [Texte imprimé] / par H. Poincaré.* s.n, S.l, 1889.

[27] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *J. Chem. Theory Comput.*, 2008.

[28] J. L. Knight and C. L. Brooks III. λ-dynamics free energy simulation methods. *J. Comput. Chem.*, 30(11):1692–1700, 2009.

[29] B. Kohnke, T. R. Ullmann, A. Beckmann, I. Kabadshow, D. Haensel, L. Morgenstern, P. Dobrev, G. Groenhof, C. Kutzner, B. Hess, et al. Gromex: A scalable and versatile fast multipole method for biomolecular simulation. In *Software for Exascale Computing-SPPEXA 2016-2019*, pages 517–543. Springer, Cham, 2020.

[30] X. Kong and L. Brooks III, Charles. λ-dynamics: A new approach to free energy calculations. *J. Chem. Phys.*, 105:2414–2423, 1996.

[31] C. Kutzner, R. Apostolov, B. Hess, and H. Grubmüller. Scaling of the GROMACS 4.6 molecular dynamics code on SuperMUC. In M. Bader, A. Bode, and H. J. Bungartz, editors, *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, pages 722–730. IOS Press, Amsterdam/Netherlands, 2014.

[32] C. Kutzner, D. van der Spoel, M. Fechner, E. Lindahl, U. Schmitt, B. de Groot, and H. Grubmüller. Speeding up parallel GROMACS on high-latency networks. *J. Comput. Chem.*, 2007.

[33] T. J. Lane, D. Shukla, K. A. Beauchamp, and V. S. Pande. To milliseconds and beyond: challenges in the simulation of protein folding. *Curr. Opin. Struct. Biol.*, 23(1):58–65, 2013.

[34] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and 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*, pages 1–12. IEEE, 2009.

[35] P. E. Lopes, O. Guvench, and A. D. MacKerell. Current status of protein force fields for molecular dynamics simulations. In *Molecular modeling of proteins*, pages 47–71. Springer, 2015.

[36] L. Monticelli and D. P. Tieleman. *Force Fields for Classical Molecular Dynamics*, pages 197–213. Humana Press, Totowa, NJ, 2013.

[37] J. Nicolas, K. Gubbins, W. Streett, and D. Tildesley. Equation of state for the lennard-jones fluid. *Molecular Physics*, 37(5):1429–1454, 1979.

[38] C. Niedermeier and P. Tavan. Fast version of the structure adapted multipole method–efficient calculation of electrostatic forces in protein dynamics. *Molecular simulation*, 17(1):57–66, 1996.

[39] M. Patra, M. Karttunen, M. T. Hyvönen, E. Falck, and I. Vattulainen. Lipid bilayers driven to a wrong lane in molecular dynamics simulations by subtle changes in long-range electrostatic interactions. *The Journal of Physical Chemistry B*, 108(14):4485–4494, 2004.

[40] F. Paul, C. Wehmeyer, E. T. Abualrous, H. Wu, M. D. Crabtree, J. Schöneberg, J. Clarke, C. Freund, T. R. Weikl, and F. Noé. Protein-peptide association kinetics beyond the seconds timescale from atomistic simulations. *Nat. Commun.*, 8(1):1–10, 2017.

[41] D. Potter, J. Stadel, and R. Teyssier. PKDGRAV3: Beyond trillion particle cosmological simulations for the next era of galaxy surveys. *Comput. Astrophys. and Cosmology*, 4(1):2, 2017.

[42] W. Qiu-Dong. The global solution of the n-body problem. *Celestial Mechanics and Dynamical Astronomy*, 50(1):73–88, Mar 1990.

[43] C. Sagui and T. Darden. Multigrid methods for classical molecular dynamics simulations of biomolecules. *The Journal of Chemical Physics*, 114(15):6578–6591, 2001.

[44] T. Takahashi, C. Cecka, W. Fong, and E. Darve. Optimizing the multipole-to-local operator in the fast multipole method for graphical processing units. *International Journal for Numerical Methods in Engineering*, 89(1):105–133, 2012.

[45] C. A. White and M. Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *The Journal of Chemical Physics*, 105(12):5061–5067, 1996.

[46] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[47] R. Yokota, L. A. Barba, T. Narumi, and K. Yasuoka. Petascale turbulence simulation using a highly parallel fast multipole method on gpus. *Computer Physics Communications*, 184(3):445–455, 2013.

[48] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, and K. Yasuoka. Fast multipole methods on a cluster of gpus for the meshless simulation of turbulence. *Computer Physics Communications*, 180(11):2066–2078, 2009.