

MultiXscale

Exascale pathways for performance portable electrostatics solvers

MultiXscale Deliverable 2.2
Deliverable Type: Report
Delivered in December, 2023

MultiXscale
EuroHPC Centre of Excellence for
Multiscale Modelling



Co-funded by
the European Union



Acknowledgement

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement No 101093169.

Disclaimer

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU). Neither the European Union nor the granting authority can be held responsible for them.

Project and Deliverable Information

Project Title Project Ref. Project Website EuroHPC Project Officer	MultiXscale: EuroHPC Centre of Excellence for Multiscale Modelling Grant Agreement 101093169 https://www.multixscale.eu Dr. Linda Gesenhues
Deliverable ID Deliverable Nature Dissemination Level Contractual Date of Delivery Actual Date of Delivery	D2.2 Report Public Project Month 12(31 st December, 2023) 28 th December, 2023
Description of Deliverable	Report on exascale pathways for performance portable electrostatics solvers and further plans

Document Control Information

Document	Title:	Exascale pathways for performance portable electrostatics solvers
	ID:	D2.2
	Version:	As of December, 2023
	Status:	Accepted by Steering Committee
	Available at:	https://www.multixscale.eu/deliverables
Review	Document history:	Internal Project Management Link
Review	Review Status:	Reviewed
Authorship	Written by:	Godehard Sutmann(FZJ)
	Contributors:	
	Reviewed by:	Matej Praprotnik (NIC), Alan O'Cais (UB)
	Approved by:	Matej Praprotnik (NIC)

Document Keywords

Keywords:	MultiXscale, HPCI,
-----------	--------------------

28th December, 2023

Disclaimer: This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements.

Copyright notices: This deliverable was co-ordinated by Godehard Sutmann¹ (FZJ) on behalf of the MultiXscale consortium with contributions from

. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit:

<http://creativecommons.org/licenses/by/4.0>



¹g.sutmann@fz-juelich.de

Contents

1	Introduction	2
1.1	Scalability	2
1.2	Portability	2
2	Performance Portability Considerations	4
2.1	Software products for Portability	4
3	Methods and Libraries for Electrostatics Computations	7
3.1	ScaFaCoS	7
3.2	Short and Long Range Interactions	8
4	Steps to provide a performance portable electrostatic library	10
5	Cooperation and Collaboration	11
6	Outreach and Dissemination	12
7	Conclusions	13

List of Figures

1	Measured runtime of the Ewald summation method on five different architectures.	5
2	Rescaled performance of the Ewald summation method on five different architectures.	5
3	Efficiency of the Ewald summation method implemented in Kokkos using five different architectures relative to the maximum achievable performance.	6

Executive Summary

To enable simulation codes to run efficiently on exascale architectures it is not sufficient to only consider porting and optimisation effort for existing software. The complexity of heterogeneous hardware together with a large total number of compute units requires additional effort define (and achieve independence) between tasks in simulation codes and to adjust task distribution over compute units.

For the case of electrostatic computations which are based on variants of the Ewald summation method, coarse grain task division can be achieved by considering short- and long range contributions to the electrostatic potential and forces separately. Given different performance characteristics between real space and Fourier k-space parts, multi-domain partitioning can be applied together with adjustment of relevant parameters of the Ewald method to achieve a balance of computational load between independently running computational partitions. Optimisation of operations, performed on k-space and real space parts is essential in order to achieve good load balancing between real- and k-space part partitions.

Due to the bandwidth limited performance characteristics of the FFT, the k-space partition will be much smaller than the partition on which the real space contribution is computed. The Fourier part has to consider efficient and flexible implementations of FFT libraries, running on CPUs and GPUs of different vendors.

The real space part needs additional flexibility to allow for a possible integration into the main driver simulation code. This is considered as an option if an optimised version of a function for short range interactions exists into which the electrostatic short range part could be integrated. Performance portability is essential for software which is designed for new HPC architectures, which makes it sustainable and flexible.

Performance portability can be achieved by coupling devices and using offloading of compute kernels based on, e.g., OpenMP, OpenACC, Kokkos or Raja. Based on preliminary work on an Ewald summation method, it is suggested to compare different software providing frameworks for offloading, in order to decide about the best approach. To provide software which can be used by a broader community the product will be integrated into a library.

Cooperation will be established with both internal and external project partners. Internally, expertise from WP2 partners will provide feedback and specific needs for functionality and details for code coupling. Cooperation with partners from WP1 will be essential for a stable software stack on pre-exascale and exascale machines and a seamless transition between architectures. External expertise include a cooperation with the POP CoE in an early stage of the testing phase of the electrostatic solver library to detect and analyse possible bottlenecks in the implementation and to achieve a high level of code optimisation. Dissemination of the library will start with hands-on tutorials for a broader scientific community followed by tutorials and workshops.

The presented report reflects plans and started activities of the MultiXscale partner from FZJ to enable electrostatic solvers being integrated efficiently into particle simulation codes.

1 Introduction

The target of this report is to characterize the steps to be taken to port software of time intensive simulation codes to modern high performance computing architectures. As a special focus we will consider architectures which are installed as pre-exascale architectures in Europe and will also consider features foreseen for exascale architectures in Europe, e.g. the Jupiter machine in Jülich. The report mainly considers examples coming from applications in fields such as soft matter physics and biophysics, as these are relevant for the Center of Excellence MultiXscale. However, algorithms considered here may have much broader scope and are also relevant for, e.g., Plasma Physics or Astrophysics applications. Therefore, scalability issues will be considered beyond the specific scope of MultiXscale. When discussing electrostatic solvers, we are considering methods which are designed to solve the Poisson equation. For many applications, this must also include consideration of different boundary conditions. As a consequence of this consideration, formulations for solvers have to be adjusted, which in most cases also includes algorithmic modifications.

Although the problem of solving the potential and the force at the position of a particle has been solved in principle for many practical applications, the development of efficient methods for big particle systems had a long evolution starting with the Ewald summation method in the early 20's of the last century [1, 2]. The central issue of solving the problem is (i) the conditional convergence of the summation and (ii) the internal quadratic numerical complexity with respect to number of particles, which makes the solution very expensive for upscaled system sizes. Modern methods have overcome this complexity and reduced it $O(N\log(N))$ or even $O(N)$ by taking advantage of FFT's, multipole expansions, wavelet expansions or multigrid hierarchical methods. Although the complexity can be reduced, two central questions still exist, which are related to the pre-factor and the scalability of the methods. Scaling them up to large numbers of processors comes with bottlenecks, related to collective communication behavior, which is necessary for sorting particles or to transpose computational meshes.

1.1 Scalability

An important issue to be discussed is the fact that the desired scalability, and hence the implied meaning of *scalability*, may be domain dependent. Not every community needs the weak scaling approach where the system of interest is enlarged to study, e.g. upscaling size effects. Domains like life sciences or soft matter, where the size of systems is intimately coupled to timescales, rather need efficient strong scaling, i.e. reducing computational density on each compute process to enable simulation of larger time scales. It is observed for many systems that when increasing the system size, the inherent timescale is non-linearly increased (e.g. increasing relaxation and diffusion time with $\mathcal{O}(N^{3/2})$ or $\mathcal{O}(N^2)$ depending on the inclusion of hydrodynamic interactions), i.e. for perfect weak scaling the target system size could be reached but with a non-linear increase in compute time. Therefore it has been discussed to foster also ensemble-level parallelism, which has a potential to play an increasing role in parallel molecular dynamics and statistical physics. This implies, e.g., large sets of systems with identical thermodynamic conditions but initialized with different initial conditions in velocities or coordinates. In this regard, the Copernicus ensemble framework has been developed to provide the possibility to scale up to ten-thousands of simulations in a parallel environment [3] which has been integrated into the Gromacs workflow engine [4]. Therefore, scalability has to be discussed in a broader context:

- Strong scaling: shortening, e.g., timescales for a given system size (e.g. present in protein folding)
- Weak scaling: increasing system size to study large scale phenomena (e.g. material sciences)
- Ensemble simulations: obtaining statistical evidence within a large set of similar systems of comparable size

Therefore, it will not only be important to develop software which can scale to the largest possible number of processes, but also to have software which can be coupled to different codes and can be ported to different architectures. This favors a modular software design and library solutions, from which several codes can profit and which facilitates maintenance and sustainability.

1.2 Portability

One of the central questions for the era of new HPC architectures is portability between architectures of different type. Most simulation codes have a long history and have originally been developed for traditional CPU architectures. Although variations come with, e.g., vectorization or parallelization, the underlying programming model had been stable. This changed with the advent of accelerators, in particular GPUs. Some experience had been gained with "alternative" approaches to accelerate computations, e.g. FPGAs or streaming architectures but this has not (yet) been established as a standard in HPC [5, 6]. This is in contrast to GPU accelerators, which nowadays are the main performance drivers of cutting edge HPC machines. Although the GPU's now form a major part of the HPC landscape, they are still a challenging target for code development. This is due to an unusual data layout and unique native language paradigms, both of which pose compatibility issues for traditional codes, typically written in C, C++ or Fortran. Some

support has been provided to address the compatibility layer between CPU's and GPU's via directive based additions, like OpenACC or OpenMP, and language based frameworks like Kokkos or Raja. These facilitate porting to some extent, but also introduce another level of complexity to codes.

This report will provide a plan of how existing implementations for solving the electrostatic problem have to be extended to provide the basis for being ported to exascale machines, how the performance portability issue has to be considered and how task decomposition in such codes might be adapted to allow for overall scalability of the codes. The solution of the electrostatic problem in particle codes definitely plays a dominant role for the overall consumption of compute time. However, for the final scalability of a code all parts/tasks of the simulation code or workflow have to be considered. In this report, our focus is placed on the electrostatic part, which nevertheless has to be considered together with the task of solving short range interactions, as all fast solvers split the total work into short- and long-range contributions. As will be discussed, this splitting not only is a technical necessity, but also enables optimization between parts, which show different scalability behavior.

2 Performance Portability Considerations

Due to the increase in heterogeneous computer systems, porting software between different platforms has become more and more important. In particular, the transition from CPU to GPU architectures has posed a challenge for many codes.

Both hardware vendors and developers of parallel libraries offer *offloading* solutions to port software between architectures. This, however, does not guarantee that the performance on the accelerator platform reaches the relative performance, which has been achieved on the CPU side. Possible reasons include missing optimization on the accelerator side or different memory access patterns, i.e. non-optimal or non-adjusted data structure layout.

To realise the transition from CPU to GPU, a change in the layout of data structure might be necessary, e.g. switching from an AoS (array-of-structure) type particle list to a SoA (structure-of-arrays) type list. Usually SoA style particle lists show better properties in streaming and cache usage (on a GPU). Therefore, porting code without the adjustment of data structures often results in loss of performance.

The alternative to offloading kernels via tools is the specific design of kernels in a supported language of the acceleration platform, e.g. CUDA for the Nvidia GPU's. This approach, however, makes it necessary to rewrite portions of the code when changing to other types of GPU's and therefore restricts porting to a specific vendor.

Performance portability is key to use different accelerator platforms efficiently. A simple definition of performance portability is that the same source code will run *productively* on a variety of different architectures [7]. However, there is a debate in literature on how to actually quantify performance portability [8, 9, 10].

Our aim is to take a more generic approach, where the electrostatic solvers can be used on diverse hardware architectures. Historically, OpenCL [11] was established as alternative to CUDA providing a programming environment to target different types of architectures. While OpenCL is at present not as prominent as in the past it is still part of developments such as ROCm [12] from AMD.

2.1 Software products for Portability

Examples of software products, enabling the transition from CPU to GPU or parallel use include

- OpenMP [13] has been invented as a shared memory parallel programming paradigm. It has been further developed to allow off-loading to accelerator devices. It follows the so-called host-device model, where the host creates the data environment and maps data to this environment on the device. Compute kernels within “target” regions are off-loaded to the target architecture and operations between OpenMP directives are executed on the device, from where data are back-transferred to the host. OpenMP host device model allows for both C/C++ and Fortran code to be off-loaded. Using OpenMP clauses, multi-level parallelism can be achieved (e.g. combination of `omp teams` and `omp parallel do`).
- OpenACC [14] is an initiative of CRAY, CAPS, Nvidia and PGI to enable existing C/C++ and Fortran codes to run on heterogeneous CPU/GPU systems. With simple compiler directives in the source code, compute kernels can be off-loaded to a GPU. This enables computations on the GPU but does not guarantee maintaining performance. Optimization is available at compile time, but data layout is not automatically transposed from CPU to GPU. This is one reason for the performance loss between native code, designed for the GPU and offloaded kernels with CPU adjusted data layout.
- HIP [15] is a heterogeneous compute interface from AMD. It consists of a C++ runtime API and kernel language that allows to develop code for Nvidia and AMD GPU's. It allows for coding in C++ code and to use many modern features such as templates or lambdas. The tool HIPIFY allows to port CUDA code from Nvidia cards to AMD cards, providing an automatic translation from CUDA into HIP C++ to be run on AMD cards. Effort is made to establish a common code base for AMD and Nvidia cards. However, it is not guaranteed that the performance of the CUDA code is fully maintained.
- Raja [16, 17] and Kokkos [18, 19] are both C++ based products. Both provide many backends, so that code can be run on different target devices. Raja is developed at LLNL and consists of a collection of C++ abstractions. On the other hand, Kokkos is developed at Sandia NL and provides a C++ framework. It has introduced so-called views which map arrays from the program to nearly optimal data layout on the target device. This allows a rather seamless transition from architecture to another, without the need to architecture dependent data layout on code level. This strongly facilitates coding and maintaining a code. For Kokkos, portability relies on the up-to-date version of Kokkos, i.e. whether the target architecture has been included into the Kokkos data view (however, it is also possible for the programmer to experiment with and to adjust the data layout in the code under development).

For compute kernels with a low algorithmic complexity, different implementations, e.g. OpenMP, OpenACC and Kokkos, can be directly compared. In a first implementation phase the different approaches will be compared to each other in a testbed environment for the case of a typical compute kernel.

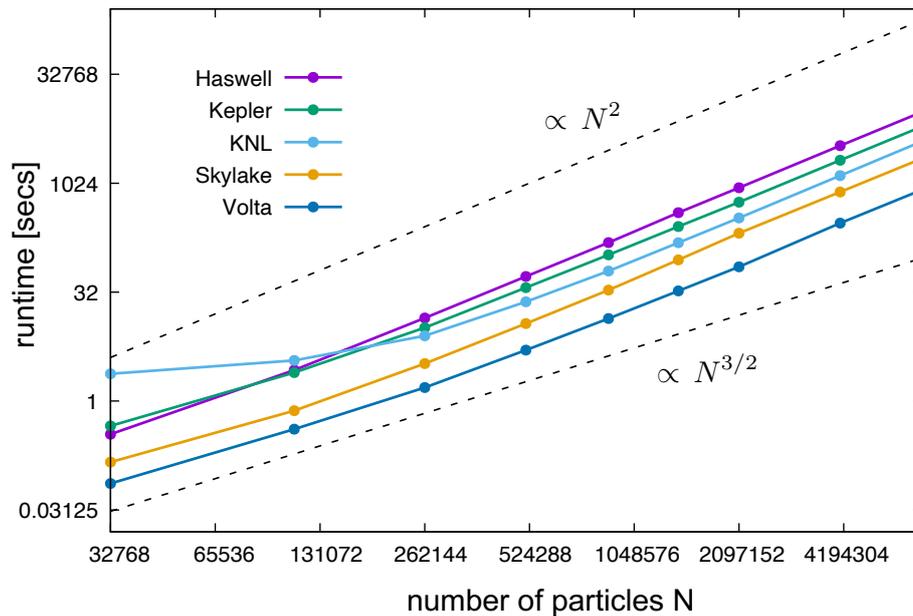


Figure 1: Measured runtime of the Ewald summation method on five different architectures.

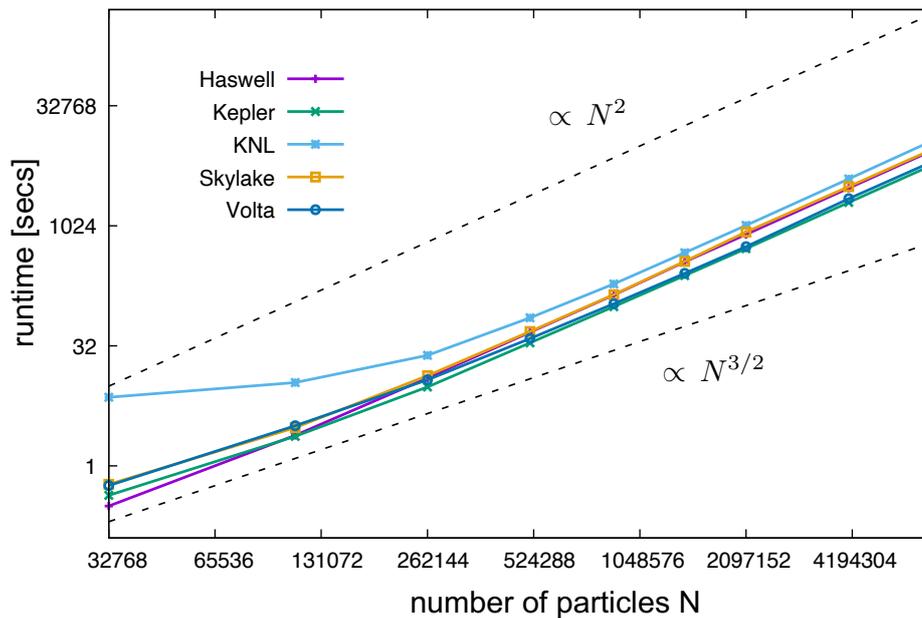


Figure 2: Rescaled performance of the Ewald summation method on five different architectures.

For the case of Kokkos, some experience has already been gained for the computation of short range interactions for the case of a traditional Ewald summation as well as for an example of particle based hydrodynamics computation in the formulation of Multi-Particle Collision Dynamics (MPCD) [20, 21]. The scientific case of Ewald summation has been tested on a number of platforms (cmp. Fig. 1) [Halver2020a,Sutmann2020a]. Since Kokkos is an on-node parallelization environment, performance has been tested for single-node only with a modification of particle number. As can be expected (i) the runtime increases with the number of particles (here a quadratic increase since all particles in the central simulation box interacted with each other); (ii) depending on the architecture performance capability the runtime differs between the architectures.

In order to check the specific issue of performance portability, the number of performed operations on each architecture has been measured and put in relation with maximum achievable performance which could have been obtained

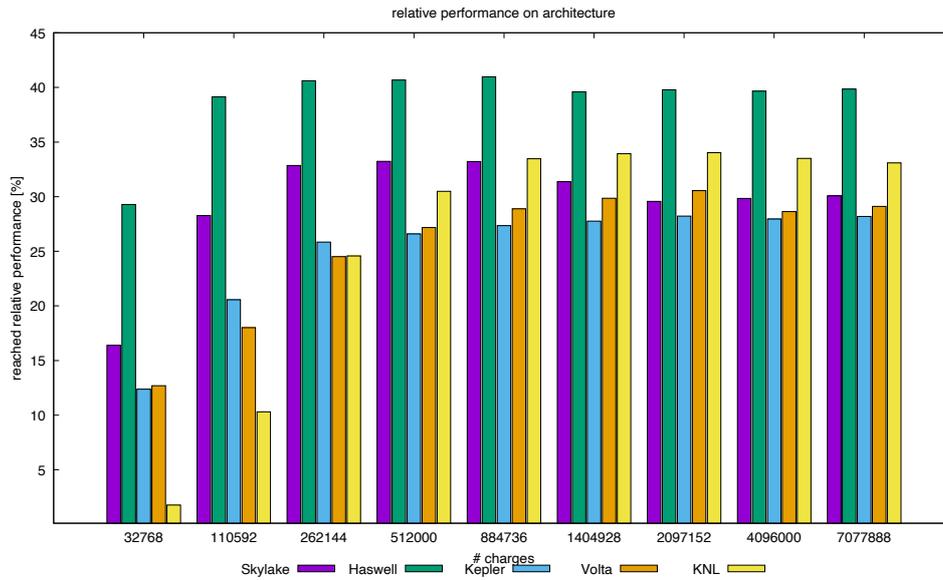


Figure 3: Efficiency of the Ewald summation method implemented in Kokkos using five different architectures relative to the maximum achievable performance.

if operations could be performed without memory or runtime overhead. The resulting comparison of this rescaled performance is shown in Fig. 2 from where it is nicely seen that the use of Kokkos enables performance portability in terms of achievable performance on a given architecture within a narrow band of performance differences. In a different representation this is also shown as relative performance compared to the maximum achievable performance in Fig. 3. Although a direct comparison between Kokkos and both OpenMP and OpenACC for a similar case study is not yet performed, a first choice for the portability work in Task 2.3 of WP 2 will be to start with the Kokkos framework.

3 Methods and Libraries for Electrostatics Computations

In many simulations, electrostatics calculations are one of the more expensive computational parts. When using a naive approach they scale quadratically with the number of charges in the system, i.e. as $\mathcal{O}(N^2)$. Since this complexity hinders scalability to large systems and introduces prohibitive computational cost, considerable effort has been devoted to the development of fast methods, which all rely on a separation of space in terms of long-range and short-range contributions. Promising methods for higher scaling are the Fast Multipole Method, variations of the Ewald method using FFT's for the far field, multigrid method or wavelet based methods. While FMM and multigrid promise to scale linearly with number of charges, the other methods exhibit computational complexity of $\mathcal{O}(N \log(N))$. Although a computational complexity of $\mathcal{O}(N)$ looks promising for large systems, parallel scalability of these methods has not yet been shown to have a clear advantage. Differences and preferences of the methods can be found in advantages for specific applications, e.g. Monte Carlo techniques, where only energy changes upon single particle changes are to be computed. Since the majority of applications in physics and materials science, which need the electrostatics calculations, apply system dynamics or global changes between iteration steps, we will not focus on such special applications.

3.1 ScaFaCoS

For electrostatic calculations, boundary conditions play an important role for the physical setup. Several variants of FMM, P3M and wavelet methods have been developed to distinguish between periodic, partial periodic, open and finite-geometry boundary conditions. Therefore, there is not only one method which has to be considered if one deals with porting of electrostatics to new architectures. In view of this diversity, the ScaFaCoS library [22] has been developed, which provides a number of different solvers for the electrostatic problem. Originally the development had two goals: (i) make methods comparable in a common benchmark and development environment and test performance and scalability under controllable and comparable conditions; and (ii) compare methods, which might perform differently for different simulation conditions, e.g. number of charges or different simulation requirements, e.g. level of accuracy. This also included the goal to include method variants dealing with different boundary conditions.

A comparison of the scalability of the ScaFaCoS methods has revealed that the implemented methods had different performance properties [23]. From those properties a clear superiority of a certain method could not be identified. All included methods have in common the separation of the computation into long- and short-range contributions. For the comparative study, it has not been taken into account whether different scalability properties of these two parts might exist. Rather the short and long-range parts were setup for a single or small set of processors with the requirement of optimizing the total runtime over the benchmark specification. From these initial set of parameters, the upscaling was performed. Especially for the methods including FFTs, this is certainly not the optimal choice. Therefore, a porting of the methods also has to take into account an optimization process to balance the work in the short- and long-range parts not only on small processor counts but to take into account the high demand of communication of parts of the computation, e.g. the FFT. This calls for a hybrid partitioning of the tasks in the computations, i.e. performing computation of bandwidth limited parts on a smaller subset of resources. Since the porting of the methods have to include GPU architectures, the performance optimization needs a strategy to partition resources for different tasks.

The existing ScaFaCoS library is already integrated into the ESPResSo code [24, 25] of the MultiXscale partner in Stuttgart, where the P3M method is applied for most applications including electrostatics interactions between particles. It has also been integrated into the LAMMPS simulation engine [26, 27, 28]. The FMM code from the library has been further developed in the context of the Gromacs code [29], where the special need for lambda-dynamics was required, which needs energy changes of single groups of particles, i.e. local changes, and does not require in each step the update of the full system energy [30]. Since the FMM is already being developed further in a different context, the main focus of the porting approach within MultiXscale will be on the P3M method and variants for boundary conditions and possibly on wavelet methods. The wavelet methods in particular have been demonstrated to capture both the field of high accuracy computations and achieve high scalability. These methods were developed by Gödecker and collaborators [31, 32, 33, 34, 35], and have been both integrated into the DFT code BigDFT [36, 37] and provisioned as stand-alone package to be included into existing simulation codes with small adjustments.

The described methods are available in different development stages. The methods implemented into ScaFaCoS have C, C++ and Fortran90 interfaces. The wavelet methods available from the BigDFT or the GitLab repository are in Fortran90. C++ versions have been realized but are not yet accessible as open source. This also includes a GPU ready version of Poisson solvers for various boundary conditions, including open and periodic boundary conditions [38]. Therefore, addressing porting also implies a modernization of ScaFaCoS and the methods included therein. As a result, it is important to identify first a set of methods which are serving the MultiXscale community and have the potential to reach out to a wider community so as to focus development effort where it is likely to have the most impact.

From the point of view of applications in the MultiXscale context, the P3M method is most promising as it has been applied to a large set of applications in the context of soft matter physics and complex system simulations. In the context of P3M, various modifications were developed. This includes the electrostatic layer correction (ELC) [39], which takes into account restricted geometries with plane walls in one or two dimensions. In principle this allows also for open boundaries. Other modifications include the introduction of electrostatic shadowing which includes modified dielectric properties in the system. Therefore, the starting point for the porting approach will consider the P3M method, as it is currently used in the ESPResSo code. To achieve highest efficiency and flexibility, the method is modularized and prepared to be included into an integrated library.

3.2 Short and Long Range Interactions

All of the mentioned candidate methods rely on splitting the electrostatic problem into a short- and long-range part. The short-range part can be computed by standard means of short-range molecular dynamics methods including setup of neighbor lists and computing explicit pair-pair particle interactions within a cutoff region. Short-range real space contributions can be mapped to a standard domain decomposition, where particle information is imported/exported from neighbor domains, i.e. having local range of interactions. This is usually implemented by point-to-point communications between processes. On a multi-GPU system this is consequently translated to inter-GPU point-to-point communication via ROCm [12] or GPUdirect [40].

Since computation of short-range interactions between particles is common for all MD community codes, implementation has received much attention, resulting mainly in highly optimized routines. Therefore the short-range part of electrostatic interactions is likely to be computed in the MD driver program. In this case a library solution has to provide the proper functions, which are required by the Poisson solver. However, computing short-range interaction in the MD code comes with restrictions concerning the size of the short-range part, which has to be considered as an optimization parameter for balancing long- and short-range contributions. If the cutoff radius in the program is adjusted to sizes which are typically several particle diameters large, this is not necessarily true for the Coulomb short-range part. Increasing this radius in the program would need to adjust the interaction range for different types of interactions which might introduce another overhead or require an independent implementation of the short-range part in the program. Therefore, a library solution will also need to offer an implementation of the computation of the short-range contributions. This has as a consequence, that parts of the library are called simultaneously on different domain partitions in the case where the long-range part is computed on a subset of CPUs/GPUs.

For short-range part of electrostatic interactions, lists provide an efficient way to reduce the number of distance computations between particles. Lists can be constructed in different ways, e.g., Gromacs [29] chooses the way to define small clusters, which are interacting and which can be mapped profitably onto SIMT, SIMD-based CPU and GPU. In this way, the possibility for vectorization is inherently included in the algorithm. Similarly, offloading via OpenMP or OpenACC can also profit and it has to be investigated whether Kokkos would further profit from introducing data locality via cluster interactions.

Similar to clusters (which have to be constructed during runtime) is the introduction of containers, holding a certain number of particles. They can be defined similar to linked lists, making it fast to sort particles geometrically. Containers can host particle data locally, forming a data local mesh of particle containers. This can be used profitably to implement similar data locality and data reuse patterns as in mesh based computations. An advantage is that a static mapping of containers can be performed, which also allow static patterns for data access, similar to mesh-based operations. Since containers might only be filled partially they increase overall memory demand, which introduces bottlenecks in memory management. It has to be evaluated which short-range implementation performs best and whether it might be dependent on the underlying architecture.

In contrast, FFT works on a global mesh, which has to be transposed after each transform in a cartesian direction. This involves global communication, which dominates the FFT calculation for large partitions, i.e. makes it bandwidth limited. Therefore, the runtime can be optimized by choosing a hybrid approach for partitioning resources, where part of the resources works the main part of the simulation, including the real space short-range contributions, while another smaller part of resources works on the FFT part. In principle, for a heterogeneous CPU/GPU system, the FFT part could be run on the CPU partition. This partitioning is certainly a very important step to reach at a balanced distribution of runtime together with a minimization of a communication cost function. Therefore, extensive benchmarks together with runtime models will provide information about mapping of programs onto the CPU/GPU resources or on partitioned GPU's.

Effort has been spent in developing new FFTs or improving runtime behavior and scalability [41, 42]. A subset of available FFT implementations is able to run on GPU's. Since the current project has not the goal to design and implement a new version of FFT, it is important to find the best suited version in terms of minimal communication overhead, fast computation and good scalability. An important step for building a strategy of resource partitioning and implementation is therefore the scalability characteristics of available FFT methods as function of mesh size and

CPU/GPU resources.

From a functional perspective, the larger part of the resources is dedicated to real space part calculations and to the preparation of the charge density distribution on the mesh. This is also a short-range contribution where the influence region of a mesh point to neighbored charges depends on the range of finite support of the method (often third-order cardinal splines or truncated Gaussian functions). This work is linearly scaling with number of charges and has no dominant contribution. The operations have generally small arithmetic intensity, so that the operations are memory bound. Independent of the performance portability framework which will be used, performance has to be monitored and possible optimizations, e.g. loop tiling or changes in memory access patterns, can be performed and support the offload tool. To support local memory access and reuse, short-range interactions could profit from sorting particles according to, e.g., space-filling curves. Although there is some overhead included, performance gain might over-compensate this additional work. Therefore, sorting routines, including space-filling curves will provide the basis for locality of particle coordinates in memory.

With respect to the Fourier space part of the computation, the charge-mesh is transferred to the domain partition, where the FFT is performed. It is assumed that this is a non-local memory transfer, where communication is performed with GPU-aware MPI, e.g. ROCm for AMD and GPUdirect for Nvidia GPU's. The FFT calculation is then carried out in parallel to the real space part. As mentioned, a balanced distribution of work between real- and Fourier-space part is required for good level of load balancing. To be most flexible in optimizing resource utilization, a repartitioning should be possible. This however, has to be coordinated with the main driver code and therefore cannot be fully detailed here. Nevertheless, implementations of P3M methods and the like allow to adjust parameters, like the k-space cutoff maximum wavenumber, the real space cutoff radius or the width of the continuous charge distribution. A suitable choice of these parameters can then allow for runtime optimization and balancing the relative work between Fourier- and real-space parts [43, 44, 45, 46].

The API for inclusion of an FFT library is to be designed in a flexible way so that the underlying code can profit from the currently best performing FFT version. It has also to take into account that the FFT should be executable on both CPU and GPU. There are only a few implementations, until now, which provide this flexibility:

- HeFFTe: is the only library that provides support for distributed-memory systems with AMD and Intel GPU's. Backends involve rocM for AMD and oneMKL for Intel. It has its own backend solution [47] which avoids licensing issues but can also make use of FFTW or vendor specific single process implementations.
- AccFFT: allows distributed computing on CPUs and GPUs, implementing a hybrid message passing via MPI and CUDA.
- FFTE: has GPU support but with a restricted compiler support (PGI), which limits its use cases in view of licensing issues. As backend it uses cuFFT [48] and FFTW [49].

In addition to splitting short-range and long-range interaction parts of the Coulomb solver, it has to be clarified, whether single precision calculations can be performed in parts of the compute pipeline, which could have both advantages for faster calculation and smaller communication volume. Single precision calculation can be combined with error reduction algorithms [50, 51] (e.g. when summing contributions onto mesh points in preparation for the FFT), thereby reducing round-off errors, making it attractive when the required accuracy is not too high.

4 Steps to provide a performance portable electrostatic library

Here we provide a summary of the actions which have to be taken to develop a performance portable framework for electrostatics calculations.

- Choice of good / best candidates of electrostatics methods, originating from ScaFaCoS library and other community developed methods.
- Benchmarks and choice of the best suited FFT
- Benchmarks of bandwidth measurements between CPU/GPU and GPU/GPU to decide on packaging data and batching FFT
- Get most efficient mapping between (MD) compute nodes and FFT nodes. Set of MD compute nodes should consist of a multiple of the subset of FFT nodes for best load distribution and avoidance of redistribution. Efficiency has to be, however, measured and optimized.
- Decide on collective vs point-to-point communication during FFT
- Characterize target systems with performance achievements of FFT's
- Optimisation of all-to-all communication, e.g. using `MPI_alltoallw`, which allows for more complex data structures and strides which has high potential for the communication steps in the transpositions.
- Setting up an interface to enable simple exchange of underlying FFT for optimal execution on target architectures
- Accessing available pre-exascale architectures and, as soon as possible, the Jupiter exascale machine
- Cooperation with the EESSI project partners to have a unique installation and software environment on the target machines
- Development of a testbed framework, consisting of application scenarios, which can be used for strong and weak scaling.
- Profiling of existing methods and identifying most demanding parts in the codes
- Evaluate performance of list- vs container-implementation for particle data management for short range interactions.
- Development of tuning electrostatic solver parameters to balance load between computations of short- and long-range interactions in splitting methods.
- Development of new library approach, which can be integrated into existing codes
- Development of load balancing strategy for task partitioning by considering minimization of communication need for bandwidth limited kernels, e.g. FFT.
- Exploring possibilities to concurrently use CPU and GPU resources and to exploit modular supercomputing
- Setup of CI environment, including
 - Gitlab repository
 - Cross compiler checks
 - Documentation
 - Cross architecture checks
- Integrating the library approach into selected codes
- Setting up a benchmark environment for testing on different architectures and archiving benchmark results

5 Cooperation and Collaboration

Besides outreach activities it is important to have feedback, exchange and support from specialists in fields not fully covered by local research groups. In particular this includes:

- Performance analysis tools
- Progress in performance portability frameworks
- Tools for software development, provision and deployment

For performance analysis tools close cooperation has to be established to the EuroHPC Center-of-Excellence POP, which has a strong record not only in analysis but also in support for performance improvement. It will be very important to follow the developments of POP with respect to provision of analysis tools for pre-exascale and exascale architectures. Further development in performance portability frameworks will be actively followed by close cooperation with core developer groups, which will also allow for constructive feedback and co-design approaches. Current cooperation includes groups from LANL, where Kokkos based library Cabana is developed, which has been applied in a recent work [52, 53]. Furthermore, developments in, e.g., OpenMP or OpenACC will be followed in view of being advantageous with respect to Kokkos (the framework, currently chosen as candidate for performance portability).

During development, testing and provision of new software, tools and environment for development, provision and deployment will profitably be used from the MultiXscale branch of the EESSI workpackages. It is important already in an early development stage to make benefit from the software provision environment, provided by the EESSI group. This will provide a transparent and seamless transition between the EuroHPC pre-exascale and exascale machines. As a result of the EESSI workpackages, the same software stack will be available on each target architecture, which will allow both the development of software within the target architecture software environment and the seamless transition from development to production stage on a number of different architectures. In addition, the cooperation will also allow for a co-design approach by providing feedback between development groups of software provision environment and scientific software developments. Furthermore, cooperation with WP6 will be fostered and that developed software for electrostatic interaction in particle systems will also provide material for training events organized within MultiXscale.

Collaboration activities

- Together with MultiXscale partner Stuttgart, software for electrostatic solvers will be tested with the MD package ESPResSo
- Codesign activities with the Stuttgart partners will be established, especially in view of library design and functionality.
- In line with project partners from WP2 (NIC), the electrostatic solver library should be coupled to other community codes, e.g. LAMMPS, and tested on different architectures.

6 Outreach and Dissemination

It is important to include a wider community into the development and testing phase of the performance ported electrostatic methods already during the development phase. This is not only important for providing new methods and tools to the community, but also to receive feedback from the a wider and inter-disciplinary audience. This will not only provide user experience on a practical level, but further allows to get into discussion about new developments in performance portable approaches, which can either enrich existing implementations or lead to alternative pathways, which might be considered as an alternative for a later release. An interesting and natural community to be included into these outreach and dissemination activities is the CECAM community, where computational scientists from different research fields come together. Electrostatic problems are relevant in both classical techniques like molecular dynamics or Monte Carlo methods, applied in Soft Matter Physics, Biophysics, Statistical Physics and Materials Science, as well as in ab initio methods, applied in reactive systems or first principles materials applications. It is also of interest to include approaches, relying on machine learning or AI, which need efficient methods for large sets of training data. For reaching out to a most broad user community base, the other CoE's will be invited to contribute or participate in improving the implementations by testing or active contributing. In this respect it is intended to have different instruments of exchange:

- **Workshops** This is intended as a platform for knowledge exchange within and across the communities. New approaches for performance portability as well as new ideas for electrostatic algorithms are discussed. This also includes topics like (i) including electrostatics into Multi-Level- and Multi-Scale-Simulations where different time and length scales are considered concurrently or; (ii) including long-range effects into ML protocols, which often rely on local structure or pattern recognition.
- **Tutorials** During tutorials the basic concepts of fast electrostatic solvers together with efficient approaches for parallelization are taught to students and postdoctoral researchers. In addition, performance portable approaches, based on, e.g., OpenMP or Kokkos are introduced.
- **Hands-on meetings** During hands-on tutorials, the developed software from WP2 on electrostatics will be provided where researchers can work together with tutors from the project on their own codes. The integrated library on performance portable electrostatics solvers will be provided in its current development stage. This will have a mutual benefit on the project and the community. On the one side the community can profit from the distribution of a developed software, on the other side the project will receive feedback on possible bottlenecks in integrating the library into existing codes, so that a continuous correction and improvement process is in place.

These kind of outreach and dissemination actions can be organized, e.g., as part of the CECAM flagship event calendar upon application, or as CECAM node events, e.g., at the CECAM-DE-Juelich or the CECAM-DE-SMSM (Mainz, Stuttgart, Darmstadt). Further outreach activities comprise presentations on external conferences and workshops, where also hands-on sessions are offered together with introductory sessions into performance portability and long range interactions.

7 Conclusions

Implementation of electrostatic long range interactions require flexibility in distributing compute kernels over heterogeneous resources, especially CPU and GPU devices. This flexibility can be achieved with adapting modern framework implementations for performance portability. It is suggested to use Kokkos as performance portability framework, as there are already promising results obtained for similar, yet different, tasks. It is suggested to also have a comparison with other kernel offload libraries, e.g. OpenMP and OpenACC, for smaller kernels to get a direct ranking. For the highest flexibility of the computations, a library approach is suggested which separates long- and short-range contributions from the calculations, in order to decide whether short range part can be covered by the driver program (e.g. MD) or should be delegated to the library. It will be important to connect to the scientific user community in order to have both the information exchange about experiences in applying the library and the feedback about performance and correctness issues. Workshops and hands-on are planned to satisfy these needs.

Acronyms Used

AI	Artificial Intelligence
ALL	A Load-balancing Library
API	Application Programming Interface
CECAM	Centre Européene de Calcul Atomique et Moléculaire
CI	Continuous Integration
CoE	Center of Excellence
CPU	Central Processing Unit
DFT	Density Functional Theory
EESSI	European Environment for Scientific Software Installations
ELC	Electrostatic Layer Correction
ESDW	Extended Software Development Workshop
ESPResSo	Extensible Simulation Package for Research on Soft Matter
EuroHPC	The European High-Performance Computing
EuroHPC JU	European High Performance Computing Joint Undertaking
FFT	Fast Fourier Transformation
FFTW	Fastest Fourier Transform in the West
FMM	Fast Multipole Method
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HeFFTe	Highly Efficient Fast Fourier Transforma for Exascale
HIP	Heterogeneous Interface for Portability
HPC	High Performance Computing
HTC	High Throughput Computing
HPDA	High Performance Data Analysis
JSC	Jülich Supercomputing Centre
KPI	Key Performance Indicators
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LANL	Los Alamos National Laboratories
LLNL	Lawrence Livermore National Laboratories
ML	Machine Learning
MPCD	Multi Particle Collision Dynamics
MD	Molecular Dynamics
MPI	Message Passing Interface
NIC	National Institute of Chemistry
NL	National Laboratories
openACC	Open Accelerators
openMP	Open Multi-Processing
P3M	Particle-Particle Particle-Mesh method
POP	Performance Optimisation and Productivity
PRACE	Partnership for Advanced Computing in Europe
ScaFaCoS	Scalable Fast Coulomb Solvers
SIMD	Single Instruction Multiple Data
SIMT	Simultaneous Multithreading
SOA	Structure Of Arrays
UB	University of Barcelona
WP	Work Package

URLs referenced

Page ii

<https://www.multixscale.eu> ... <https://www.multixscale.eu>
<https://www.multixscale.eu/deliverables> ... <https://www.multixscale.eu/deliverables>
 Internal Project Management Link ... <https://github.com/orgs/multixscale/projects/3>
g.sutmann@fz-juelich.de ... <mailto:g.sutmann@fz-juelich.de>
<http://creativecommons.org/licenses/by/4.0> ... <http://creativecommons.org/licenses/by/4.0>

References

- [1] P. Ewald, "Die Berechnung optischer und elektrostatischer Gitterpotentiale," *Ann. Phys.*, vol. 64, p. 253, 1921.
- [2] B. Luty, M. Davis, I. Tironi, and W. van Gunsteren, "A comparison of particle-particle, particle-mesh and Ewald methods for calculating electrostatic interactions in periodic molecular systems," *Mol. Sim.*, vol. 14, pp. 11–20, 1994.
- [3] S. Pronk, I. Pouya, M. Lundborg, G. Rotskoff, B. Wesen, P. Kasson, and E. Lindahl, "Molecular Simulation Workflows as Parallel Algorithms: The Execution Engine of Copernicus, a Distributed High-Performance Computing Platform," *J. Chem. Theo. Comp.*, vol. 11, pp. 2600–2608, 2015.
- [4] [Http://Copernicus.gromacs.org](http://Copernicus.gromacs.org).
- [5] C. Yang, T. Geng, T. Wang, R. Patel, Q. Xiong, A. Sanaullah, C. Wu, J. Sheng, C. Lin, V. Sachdeva, W. Sherman, and M. Herbordt, "Fully integrated FPGA molecular dynamics simulations," in *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.
- [6] D. J. and J. E. Allen, Y. Y. and W. F. D. Bennett, M. Gokhale, N. Moshiri, and T. Rosing, "Accelerators for Classical Molecular Dynamics Simulations of Biomolecules," *J. Chem. Theory Comput.*, vol. 18, p. 4047–4069, 2022.
- [7] J. Larkin, "Performance Portability Through Descriptive Parallelism," in *DOE CoE Performance Portability Workshop 2016*, 2016.
- [8] S. Pennycook, J. Sewall, and V. Lee, "Implication of a Metric for Performance Portability," *Future Generation Computer Systems*, vol. 92, pp. 947–958, 2019.
- [9] S. Pennycook and J. Sewall, "Revisiting a Metric for Performance Portability," in *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, vol. 2, 2021, pp. 1–9.
- [10] A. Marowka, "Reformulation of the performance portability metric," *Software Practice and Experience*, vol. 52, pp. 154–171, 2022.
- [11] 2016, Khronos OpenCL Working Group. The OpenCL Specification, 2.2 edition, March.
- [12] [Https://rocm.docs.amd.com/en/latest](https://rocm.docs.amd.com/en/latest).
- [13] [Http://www.openmp.org](http://www.openmp.org).
- [14] [Http://www.openacc.org](http://www.openacc.org).
- [15] [Https://rocm.docs.amd.com/projects/HIP/en/develop/index.html](https://rocm.docs.amd.com/projects/HIP/en/develop/index.html).
- [16] [Https://computing.llnl.gov/projects/raja-managing-application-portability-next-generation-platforms](https://computing.llnl.gov/projects/raja-managing-application-portability-next-generation-platforms).
- [17] D. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. Kunen, O. Pearce, P. Robinson, B. Ryujin, and T. Scogland, "RAJA: Portable Performance for Large-Scale Scientific Applications," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-CONF-788757, 2019.
- [18] [Http://kokkos.org](http://kokkos.org).
- [19] H. Edwards, C. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *J. Parallel Distrib. Comput.*, vol. 74, p. 3202–3216, 2014.
- [20] G. Gommer, T. Ihle, D. Kroll, and R. Winkler, "Multi-Particle Collision Dynamics: A Particle-Based Mesoscale Simulation Approach to the Hydrodynamics of Complex Fluids," *Adv. Polym. Sci.*, vol. 221, pp. 1–87, 2009.
- [21] G. Sutmann, C. Huang, R. Winkler, and G. Gommer, "Semidilute Polymer Systems under Shear Flow," in *IAS Series*, M. K. G. Münster, D. Wolf, Ed., vol. 3. Jülich: Forschungszentrum Jülich, 2010, pp. 287–294.
- [22] [Http://www.scafacs.de](http://www.scafacs.de).
- [23] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsels, 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*, vol. 88, p. 063308, 2013.
- [24] H. J. Limbach, A. Arnold, B. A. Mann, and C. Holm, "ESPresSo - An Extensible Simulation Package for Research on Soft Matter Systems," *Comp. Phys. Comm.*, vol. 174, pp. 704–727, 2006.
- [25] [Http://espressowiki.mpip-mainz.mpg.de/wiki/index.php/Main_Page](http://espressowiki.mpip-mainz.mpg.de/wiki/index.php/Main_Page).
- [26] S. Plimpton, "Fast parallel algorithms for short range molecular dynamics," *J. Comp. Phys.*, vol. 117, p. 1, 1995.

- [27] A. Thompson, H. Aktulga, R. Berger, D. Bolintineanu, W. Brown, P. Crozier, P. in't Veld, A. Kohlmeyer, S. Moore, T. Nguyen, R. Shan, M. Stevens, J. Tranchida, C. Trott, and S. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 217, p. 10817, 2022.
- [28] [Http://lammps.sandia.gov](http://lammps.sandia.gov).
- [29] [Http://www.gromacs.org](http://www.gromacs.org).
- [30] B. Kohnke, C. Kutzner, and H. Grubmüller, "A GPU-Accelerated Fast Multipole Method for GROMACS: Performance and Accuracy," *J. Chem. Theory Comput.*, vol. 16, pp. 6938–6949, 2020.
- [31] S. Goedecker, "Linear Scaling Electronic Structure Methods," *Rev. Mod. Phys.*, vol. 71, pp. 1085–1123, 1998.
- [32] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, "Efficient solution of Poisson's equation with free boundary conditions," *J. Chem. Phys.*, vol. 125, p. 074105, 2006.
- [33] L. Genovese, T. Deutsch, and S. Goedecker, "Efficient and accurate three-dimensional Poisson solver for surface problems," *J. Chem. Phys.*, vol. 127, p. 054704, 2007.
- [34] S. Ghasemi, A. Neelov, and S. Goedecker, "A particle-particle, particle-density algorithm for the calculation of electrostatic interactions of particles with slablike geometry," *J. Chem. Phys.*, vol. 127, p. 224102, 2007.
- [35] A. Neelov, S. Ghasemi, and S. Goedecker, "Efficient solution of Poisson's equation with free boundary conditions," *J. Chem. Phys.*, vol. 125, p. 074105, 2007.
- [36] S. Mohr, L. Ratcliff, P. Boulanger, L. Genovese, D. Caliste, T. Deutsch, and S. Goedecker, "Daubechies wavelets for linear scaling density functional theory," *J. Chem. Phys.*, vol. 140, no. 20, p. 204110, May 2014.
- [37] [Https://l_sim.gitlab.io/bigdft-doc/index.html](https://l_sim.gitlab.io/bigdft-doc/index.html).
- [38] N. /Dugan, L. Genovese, and S. Gödecker, "A customized 3D GPU Poisson solver for free boundary conditions," *Comp. Phys. Comp.*, vol. 184, p. 18151820, 2013.
- [39] J. de Joannis, A. Arnold, and C. Holm, "Electrostatics in Periodic Slab Geometries II," *J. Chem. Phys.*, vol. 117, p. 2503, 2002.
- [40] [Https://developer.nvidia.com/gpudirect](https://developer.nvidia.com/gpudirect).
- [41] M. Pippig, "PFFT - An Extension of FFTW to Massively Parallel Architectures," *SIAM Journal on Scientific Computing*, vol. 35, pp. C213–C236, 2013.
- [42] M. Pippig and D. Potts, "Parallel three-dimensional nonequispaced fast Fourier transforms and their application to particle simulation," *SIAM Journal on Scientific Computing*, vol. 35, pp. C411–C437, 2013.
- [43] D. Fincham, "Optimisation of the Ewald sum for large systems," *Molec. Sim.*, vol. 13, pp. 1–9, 1994.
- [44] H. G. Petersen, "Accuracy and efficiency of the particle mesh Ewald method," *J. Chem. Phys.*, vol. 103, pp. 3668–3679, 1995.
- [45] M. Deserno and C. Holm, "How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines," *J. Chem. Phys.*, vol. 109, p. 7678, 1998.
- [46] —, "How to mesh up Ewald sums. II. An accurate error estimate for the P3M algorithm," *J. Chem. Phys.*, vol. 109, p. 7694, 1998.
- [47] D. Sharp, M. Stoyanov, S. Tomov, and J. Dongarra, "A More Portable HeFFTe: Implementing a Fallback Algorithm for Scalable Fourier Transforms," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021.
- [48] [Https://docs.nvidia.com/cuda/cufft/index.html](https://docs.nvidia.com/cuda/cufft/index.html).
- [49] [Https://www.fftw.org](https://www.fftw.org).
- [50] B. Dmitruk and P. Stpiczynski.
- [51] —, "Improving accuracy of summation using parallel vectorized Kahan's and Gill-Moller algorithms," *Concurrency Computat. Pract. Exper.*, vol. 35, p. e7763, 2023.
- [52] R. Halver, C. Junghans, and G. Sutmann, "Kokkos-Based Implementation of MPCD on Heterogeneous Nodes," in *Lecture Notes in Computer Science*, vol. 13827, 2023, pp. 3–13.
- [53] —, "Using heterogeneous GPU nodes with a Cabana-based implementation of MPCD," *Parallel Computing*, vol. 117, p. 103033, 2023.