

Proceedings of the fourth EuroHPC User Days 2026

Scaling and performance aspects of the P3M method on multiple GPUs

Rodrigo A. C. Bartolomeu^a, René Halver^a, Godehard Sutmann^{a,*}^a*Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich, Jülich, 52428, Germany*

Abstract

Long-range electrostatic interactions constitute a major computational bottleneck in particle-based simulations, particularly when periodic boundary conditions are employed. Mesh-based Ewald methods, such as the particle–particle particle–mesh (P3M) approach, reduce the computational complexity to $O(N \log N)$ but introduce challenges related to parallel scalability, communication overhead, and efficient utilization of modern heterogeneous architectures. While highly optimized implementations exist in established molecular dynamics packages, these are typically tightly coupled to full simulation frameworks, limiting their flexibility and reuse in emerging HPC applications.

In this work, we present a performance-portable library for electrostatic solvers, providing implementations of both classical Ewald and P3M methods targeting CPU and GPU architectures. The library is designed to enable efficient execution on multi-GPU systems while maintaining portability across heterogeneous platforms. We investigate scaling and performance characteristics of the P3M method, with particular focus on the FFT-based long-range component, which is known to limit scalability. Strong-scaling experiments demonstrate that small problem sizes are dominated by communication overhead, leading to reduced parallel efficiency, whereas larger systems achieve near-ideal scaling over a wide range of GPU counts.

Our results highlight the importance of balancing computation and communication in mesh-based electrostatics and demonstrate that portable, decoupled implementations can achieve competitive performance on modern HPC systems. The presented approach facilitates integration into diverse simulation workflows and provides a foundation for further optimization of long-range solvers in exascale computing environments.

© 2026 The Authors. Published by Elsevier B.V.

This is an open access article under the CC BY 4.0 license (<https://creativecommons.org/licenses/by/4.0>)

Peer-review under responsibility of the scientific committee of the Proceedings of the fourth EuroHPC User Days 2026

Keywords: Electrostatics; FFT; P3M; Scaling; GPU computing

1. Introduction

Particle-based simulation methods, such as Molecular Dynamics (MD) and Monte Carlo techniques, are well-established tools for investigating the structural and dynamic properties of nanoscale many-particle systems [1]. In

* Corresponding author.

E-mail address: g.sutmann@fz-juelich.de

these simulations, long-range interactions, most notably electrostatic interactions, have a significant importance [2]. However, accurately computing them remains a major computational bottleneck.

Directly evaluating all pairwise interactions scales as $O(N^2)$, which is prohibitively expensive for large systems. Furthermore, naive truncation of long-range interactions introduces artefacts, necessitating the use of methods that account for all particle contributions. This challenge is commonly faced by the use of periodic boundary conditions, which are employed to reduce the effects of the system's finite size and mimic bulk behaviour. Under these conditions, the system is replicated infinitely and the resulting Coulomb sum only converges conditionally due to the long-range nature of the interactions [3].

The classical Ewald summation method addresses this issue by decomposing the electrostatic potential into a short-range contribution, which is evaluated in real space, and a long-range contribution, which is treated in Fourier space [4]. This decomposition transforms the original, slowly converging series into two rapidly converging sums, enabling controlled approximations with tunable accuracy. With appropriate parameter optimisation, the computational complexity of the Ewald method can be reduced to $O(N^{3/2})$.

Further improvements are achieved by mesh-based Ewald methods, which accelerate the evaluation of the long-range contribution using fast Fourier transforms (FFTs), reducing the complexity to $O(N \log N)$. These approaches introduce a regular grid onto which particle properties are interpolated, enabling efficient computation in reciprocal space. Prominent examples include the particle–particle–particle–mesh (P3M) method [5], the particle–mesh Ewald (PME) method [6] and the smooth particle–mesh Ewald (SPME) method [2], which differ primarily in their interpolation schemes and force assignment strategies.

The increasing use of heterogeneous high-performance computing (HPC) architectures, particularly GPU-accelerated systems, has made mesh-based Ewald methods essential for large-scale molecular simulations. Their practical performance is strongly influenced by communication patterns, FFT scalability and memory access efficiency, despite their favourable asymptotic complexity. These are all critical on modern multi-GPU platforms. Consequently, substantial effort has been devoted to redesigning these algorithms for GPU-resident execution and distributed-memory parallelism. However, achieving efficient strong and weak scaling across multiple GPUs remains challenging, particularly for the long-range mesh-based portion, where global communication and FFT operations can create bottlenecks. These challenges have driven the development of optimised implementations in widely used MD packages and supporting libraries.

In this context, mesh-based Ewald methods form a core part of the majority of contemporary molecular dynamics packages, such as GROMACS (PME) [7], LAMMPS (P3M) [8], and AMBER (SPME) [2]. These implementations have been extensively optimised for heterogeneous HPC architectures and provide highly efficient solutions for large-scale simulations. However, as they are tightly integrated into full MD application frameworks, they are challenging to reuse outside of their respective software ecosystems and difficult to adopt in custom simulation workflows or emerging multiphysics and multiscale applications. By contrast, the work presented here focuses on developing a standalone, performance-portable library for electrostatic solvers. It provides implementations of both P3M and classical Ewald summation methods designed to run efficiently on CPUs and GPUs. By decoupling these algorithms from monolithic MD codes and targeting modern heterogeneous architectures, this approach enables flexible integration and facilitates algorithmic experimentation. It also addresses the current lack of reusable, scalable electrostatics libraries for HPC environments.

2. Theoretical background

Ewald techniques are able to compute the long-range interactions contributing to electrostatic forces for many systems. Within the scope of this work, we are particularly interested in methods that concern Coulomb point charges, as they are present in both single and multi-scale particle simulations, such as atomistic and coarse-grained techniques.

Previous descriptions and implementations of such methods can be found in the literature [3]. Parallel CPU implementations are available in the Scalable Fast Coulomb Solvers (ScaFaCoS) Library, available through the [ScaFaCoS](#) project website. However, most modern architectures make the use of accelerators to offload compute intensive algorithms. Therefore, within this section, we revisit the theoretical foundation to highlight the most important considerations related to GPU implementations of the Ewald Sum and P3M algorithms.

2.1. Ewald Summation Method

The electrostatic energy between two point charges decays inversely proportional to the distance with $(1/r)$. In a system with N point charges, each with a nominal charge q_i in a cubic periodic three-dimensional space with box length L at positions $\mathbf{r}_i$ is given by:

$$E = \frac{1}{2} \sum_{i,j=1}^N \sum'_{\mathbf{n} \in \mathbb{Z}} \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{n}L|}, \quad (1)$$

with $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ and excluding the contribution for $i = j$ for $\mathbf{n} = 0$.

The potential in Equation 1 presents two significant issues: it decays slowly at large distances and varies abruptly at small distances. If only one of these issues were present, it would be relatively easy to address, as a short-range potential could be treated by a simple cutoff. This problem is addressed by the means of splitting into short and long-range contributions making use of the identity:

$$\frac{1}{r} = \frac{f(r)}{r} + \frac{1 - f(r)}{r}, \quad (2)$$

therefore, giving the possibility of circumventing the slowly decaying long range contribution of the Coulomb potential have with the straightforward summation. Here, the first term of the RHS, $f(r)/r$, is a good approximation for the short-range contribution to the total energy. The function $f(r)$ is usually chosen as a smoothly decaying function, where its decay and support can be controlled by a parameter. For the present type of application most often the complementary error function is chosen for $f(r)$. For a proper choice of parameters of the complementary error function, the second term $1 - f(r)/r$, varies slowly for all values of r , allowing its Fourier transform to be accurately represented by a limited number of $\mathbf{k}$ vectors, where $|\mathbf{k}| < k_{max}$. This enables an efficient calculation of the contribution to the total electrostatic potential in reciprocal space.

Due to the linearity of the field equations, the sums of both aforementioned terms can be performed without significant loss of accuracy. Different contributions can be split into four terms. These can be computed in parallel and are given by:

$$\begin{aligned} E^{(r)} &= \frac{1}{2} \sum_{i,j=1}^N \sum'_{\mathbf{m} \in \mathbb{Z}} q_i q_j \frac{\text{erfc}(\alpha|\mathbf{r}_{ij} + \mathbf{m}L|)}{|\mathbf{r}_{ij} + \mathbf{m}L|}, \\ E^{(k)} &= \frac{1}{2L^3} \sum_{\mathbf{k} \neq 0} \frac{4\pi}{k^2} e^{-k^2/4a^2} |\tilde{\rho}(\mathbf{k})|^2, \\ E^{(s)} &= -\frac{\alpha}{\sqrt{\pi}} \sum_i q_i^2, \\ E^{(d)} &= \frac{2\pi}{(1+2\epsilon')L^3} (\sum_i q_i \mathbf{r}_i)^2, \end{aligned} \quad (3)$$

where, $E^{(r)}$ is the real space contribution, $E^{(k)}$ is the reciprocal space, $E^{(s)}$ is the self-energy, $E^{(d)}$ is the dipole correction, the splitting function f is the complementary error function $\text{erfc}(r) = \frac{2}{\sqrt{\pi}} \int_r^\infty e^{-t^2} dt$, and the Fourier transformed density $\tilde{\rho}(\mathbf{k})$ is:

$$\tilde{\rho}(\mathbf{k}) = \int_{V_b} d^3r \rho(\mathbf{r}) e^{-i\mathbf{k} \cdot \mathbf{r}_j}. \quad (4)$$

The parameter α , referred to here as the Ewald parameter, controls the weights between the contributions from real and reciprocal space, allowing the time spent calculating each contribution to be tuned. Given that the complexity of solving particle-particle interactions in real space and the scaling behaviour of algorithms for Fourier computations differ, it is possible to find a compromise in the choices of parameters for the real and reciprocal spaces and the splitting parameter, leading to complexity of $O(N^{3/2})$. The ability to control this splitting parameter has become increasingly relevant as modular HPC systems have evolved, in which each method can be computed using different resources.

In this context, for a given level of accuracy, two parameters can be chosen from the following set: the real-space cutoff (r_{cut}), the maximum number of k modes (k_{max}) and alpha. Therefore, if the implementation supports the calculation of different contributions on several platforms or architectures, one can optimise the use of resources on modern supercomputers. Additionally, the GPGPU parallelisation paradigm enables the calculation of k -modes per particle to be performed in a more efficient manner than CPUs.

2.2. Particle-Particle Particle-Mesh Method

The P3M method maps the system charges onto a mesh, enabling the necessary Fourier transformations to be efficiently performed using a Fast Fourier Transform algorithm. Simultaneously, the basic free space Coulomb Green's function ($4\pi/k^2$) is modified to ensure that the outcome of the mesh calculation closely approximates the continuum solution. The initial step involves generating a mesh-based charge density, denoted as ρ_M , which is defined at the mesh points ($\mathbf{x}_p$) given by:

$$\rho_M(\mathbf{r}_p) = \frac{1}{h^3} \int_V d\mathbf{r} W(x_p - x)W(y_p - y)W(z_p - z)\rho(\mathbf{r}), \quad (5)$$

where h represents the mesh spacing. The charge assignment function, W , is categorized by its order, P , which indicates the number of mesh points per coordinate direction over which each charge is distributed. Specifically, W is chosen to be a cardinal B-spline, a piecewise polynomial function with a weight of one. The order P corresponds to the number of sections that comprise the function and its representation in Fourier space is:

$$\tilde{W}(\mathbf{k}) = h^3 \left(\frac{\sin(\frac{1}{2}k_x h)}{\frac{1}{2}k_x h} \frac{\sin(\frac{1}{2}k_y h)}{\frac{1}{2}k_y h} \frac{\sin(\frac{1}{2}k_z h)}{\frac{1}{2}k_z h} \right)^P. \quad (6)$$

Next, we calculate the mesh-based electric field $\mathbf{E}(\mathbf{r}_p)$ in the second step. The electric field is essentially the derivative of the electrostatic potential. However, there are several methods to implement differentiation on a lattice [9]. For this purpose, we describe the $i\mathbf{k}$ differentiation method, which involves multiplying the Fourier-transformed potential by $i\mathbf{k}$, and has been shown to be the most accurate. Using this approach, the electric field $\mathbf{E}(\mathbf{r}_p)$ can be expressed as:

$$\mathbf{E}(\mathbf{r}_p) = \overleftarrow{\text{FFT}} \left[-i\mathbf{k} \cdot \overrightarrow{\text{FFT}}[\rho_M] \cdot \hat{G}_{\text{opt}} \right] (\mathbf{r}_p), \quad (7)$$

where $\overleftarrow{\text{FFT}}$ and $\overrightarrow{\text{FFT}}$ are the backward and forward FFT respectively, and $\hat{G}_{\text{opt}}$ is the modified optimal influence function, derived by Hockney and Eastwood [5] which is written as:

$$\hat{G}_{\text{opt}}(\mathbf{k}) = \frac{\tilde{\mathbf{D}}(\mathbf{k}) \cdot \sum_{\mathbf{m} \in \mathbb{Z}^3} \tilde{U}^2\left(\mathbf{k} + \frac{2\pi}{h}\mathbf{m}\right) \tilde{\mathbf{R}}\left(\mathbf{k} + \frac{2\pi}{h}\mathbf{m}\right)}{|\tilde{\mathbf{D}}(\mathbf{k})|^2 \left[\sum_{\mathbf{m} \in \mathbb{Z}^3} \tilde{U}^2\left(\mathbf{k} + \frac{2\pi}{h}\mathbf{m}\right) \right]^2} \quad (8)$$

where

$$\widetilde{\mathbf{R}}(\mathbf{k}) = -i\mathbf{k} \frac{4\pi}{k^2} e^{-k^2/4\alpha^2} \quad \text{and} \quad \widetilde{\mathbf{U}}(\mathbf{k}) = \widetilde{\mathbf{W}}(\mathbf{k})/h^3.$$

The number of mesh points, denoted as N_M , is defined by L/h along each direction (Equation 5). In theory, N_M can be chosen arbitrarily to achieve desired accuracy. However, practical constraints arise from the efficiency of FFT libraries (such as FFTW), which support optimized algorithms for computing factors of small prime numbers. A useful mesh size formula, taking into account these constraints and the efficiency of FFTs, is:

$$N_M = 2^a \times 3^b \times 5^c \times 7^d \times 11^e, \quad (9)$$

or a power of two, as this optimization facilitates maximum efficiency in FFT computation. While finding the exact minimum mesh size necessary to meet accuracy requirements in terms of target k points is possible, it is not guaranteed to be optimal for performance. Therefore, it may be beneficial to explore nearby mesh sizes that optimize time to solution, even if this incurs slightly higher computational costs in other functions, such as charge assignment.

3. Methodology

3.1. Implementation Considerations

The presented performance-portable implementation of the P3M presented here uses the Cabana library [10] for data structures and communication algorithms, as well as providing method for the administration of the necessary grids and interfaces to the heFFTe library for the FFT computations. The Cabana library was developed within the Co-Design Center for Particle Applications (CoPA) within the Exascale Computing Project (ECP). Cabana is built on top of Kokkos [11], as well as additional, optional library dependencies. Sharing the same design philosophy, the performance-portable electrostatic library has dependencies (see Fig 1. For example: MPI for parallel distributed memory execution or heFFTe [12] for FFT computations and ALL [13] for load balancing, in order to target multiple backends.

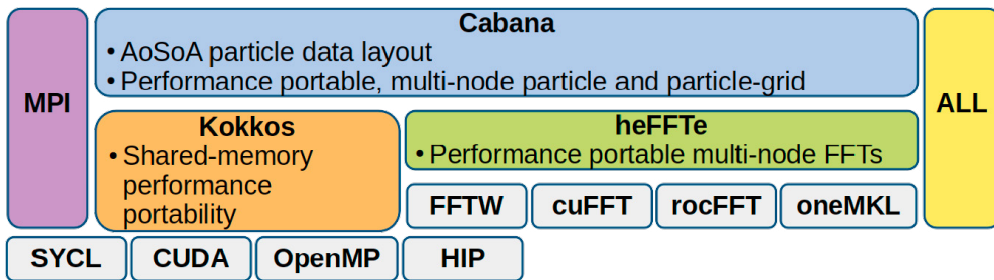


Fig. 1. Dependencies and supported backends for the current implementation of the performance portable electrostatic library.

The fact that the standard portable model for shared memory parallelism of Cabana is Kokkos, provides an intra-node parallelization model. Kokkos is suitable for particle based simulations, and the optimized conversion of data structure through the use of so-called *Views* allows for the use of optimal memory access patterns required by different parts of the electrostatics methods and many body simulations [14]. Previous work has shown, that Kokkos as a parallelization layer is suited for particle [14] and mesh-based [15] algorithms.

3.2. Benchmark setup

To explore the multi-GPU scaling capabilities of the particle-mesh based solver we have conducted computational experiments on the [MareNostrum 5 Accelerated partition](#). Nodes in this partition are equipped with two Intel Xeon Platinum 8460Y CPUs and four NVIDIA Hopper H100 64GB HBM2 GPUs each, while providing 512 GB of 4800MHz DDR5 memory, and a ConnectX-7 NDR200 InfiniBand network connection with 800 Gb/s bandwidth per core.

For this work all libraries and executables were compiled with `-O2` optimization, having the OpenMP backend enabled for host and having the CUDA backend enabled for device execution. The used compilers and libraries versions were:

- GCC 13.2.0
- CUDA 12.8
- Kokkos 4.7.2
- Cabana 0.8.0
- heFFTe 2.4.1 using the cuFFT backend

The compilers were provided by the respective modules on the test system (March 2026).

3.3. System and algorithm parameters

Knowing that the long-range contribution is the more computationally more expensive part of the electrostatic calculation as discussed in [3], our benchmarks focused on the evaluation of the elapsed compute time spent in the computation of the k -space contribution. To fairly access the impact of FFT grid sizes, the real space cutoff, r_{cut} , was fixed at 4.6172 and reciprocal space parameters were tuned to achieve an absolute accuracy of 10^{-6} . The model Cloud Wall system (Fig. 2) containing 300 particles in the base cell was replicated periodically in x , y and z directions 8, 16, 32, and 64 times, referred to as the replication size or number from here on.

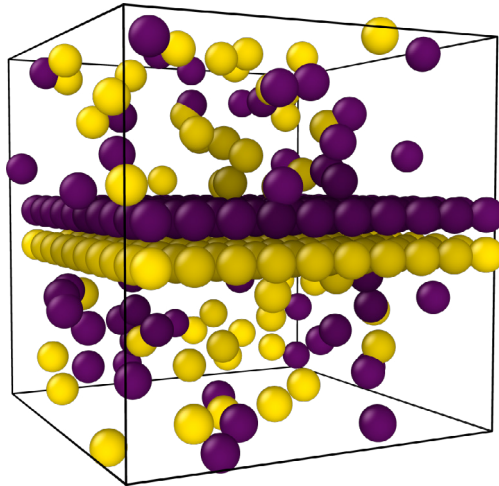


Fig. 2. Visualization of the Cloud Wall system (300 charges). The configuration was artificially created by Arnold *et al.* [3] to possess a strong long-range component. Each side has length of 10 arbitrary units.

This distribution of charged particles creates an artificial dipole. If the system is replicated, planes of opposing charges are formed. The same behaviour also occurs due to the periodic boundary conditions. Therefore, the long-range contribution is never negligible, making this configuration valuable for benchmarking the solver's accuracy.

To ensure that the predicted accuracy of the P3M method [9, 16] was achieved, results for systems up to 1228800 charges were validated through the root mean square error using a highly accurate Ewald summation tuned to yield

values close to machine precision as reference. Above this number of charges the computation of the Ewald method is considered to be prohibitively computationally expensive. As an example, for the validation of $16 \times 16 \times 16$ periodic replicas the following numbers were observed in one node using 1 GPU for long-range:

tuned Ewald values:

```
r_max: 6.30579
k_max (1D / 3D): 240 / 240 240 240
alpha: 0.871125
estimated RMS error in forces: 4.84886e-14
```

tuned P3M values:

```
r_max: 4.6172
FFT grid size (3D): 512 512 512
charge assignment order: 7
alpha: 0.846208
h * alpha: 0.26444
estimated RMS error in forces: 3.15284e-07
```

RMS potential : 4.02801676349598e-07

RMS fields: 7.64805475718963e-08 8.4353410644898e-08 3.97391771302327e-07

Time spent in run [Ewald] (avg, min, max): 2086.37708675 s 2085.901671 s 2086.535621 s

Time spent in run [P3M] (avg, min, max): 0.5853885 s 0.585388 s 0.585389 s

Where RMS indicates the absolute root mean square error with regard to per particle energy and fields. It can be seen, that the time to solution for the P3M solver is magnitudes faster than the Ewald solver ($\approx 2000s$ for the Ewald solver to $\approx 0.5s$ for the P3M solver).

Grid sizes, system details and resources used for strong scaling experiments are presented in Table 1. An important observation is that due to limitation in memory caused by the large number of FFT grid points, the larger systems can not be computed using a single computation node.

Table 1. System sizes and reciprocal space parameters of the P3M solver with fixed parameters: charge assignment order = 7, $\alpha = 0.846208$, and $r_{\text{cut}} = 4.6172$.

Replication	No. of charges	Grid size	No. of nodes	No. of GPUs for k-space
$8 \times 8 \times 8$	153600	256	1, 2, 4 and 8	1;2;3;4;6;8;12;16;24
$16 \times 16 \times 16$	1228800	512	1, 2, 4 and 8	1;2;3;4;6;8;12;16;24
$32 \times 32 \times 32$	9830400	1024	16 and 32	4;8;16;32;48;64;80;96;112;120;124
$64 \times 64 \times 64$	78643200	2048	16 and 32	32;48;64;96

The presented implementation of the P3M solver can shift MPI ranks between sub-communicators, which are used to compute the k-space and real-space contributions independently of each other. To achieve the following results, the number of processes in the k-space part was varied. A total of 64 GPUs were used for replication sizes of 8 and 16, and 128 GPUs were used for larger replication numbers (32 and 64).

4. Results

A statistical evaluation of the variation of multiple time steps using the static configuration was performed for the $32 \times 32 \times 32$ system for the presentation and post-processed P3M run times. The standard deviation remained below 3% for all numbers of GPUs. Once these values were represented graphically, it was found that they were smaller than the symbols. Therefore, error bars are omitted for clarity.

As the current study focuses on the FFT benchmark, the configuration is considered to be representative of the timings, despite the inhomogeneity of the system (which primarily affects the charge assignment routines due to load imbalance). As the library is designed to receive particle data from the calling program at each time step, particle movement is not considered to have a significant impact the current benchmarks.

Fig. 3 shows the normalised benchmark results for the Cloud Wall system with different replication sizes, as described in Section 3.3. The presented times are normalised to the smallest replication number ($8 \times 8 \times 8$) to demonstrate the similarity in scaling behaviour.

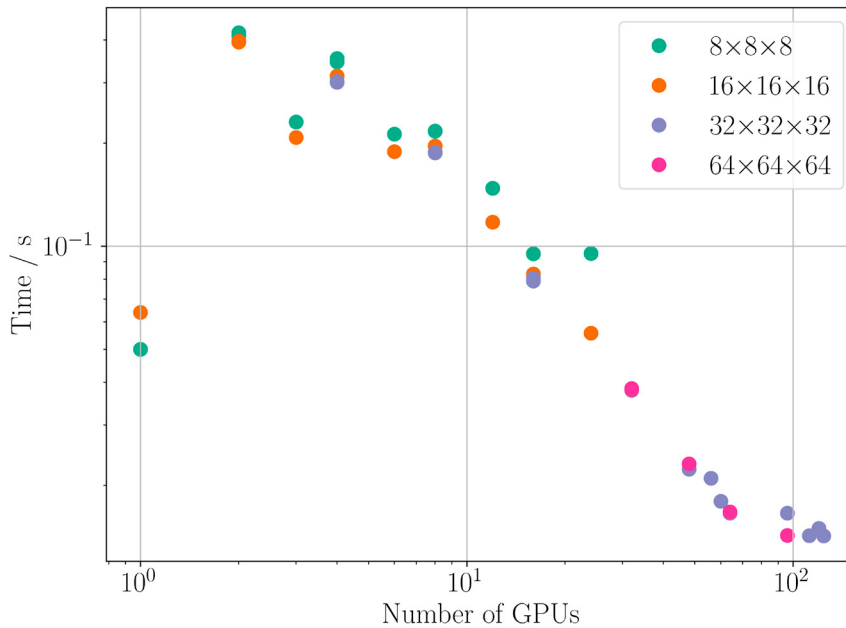


Fig. 3. Comparing strong-scaling benchmark results for different system replication numbers (8 to 64) on 1 to 128 GPUs (1 to 32 nodes) on MareNostrum5 accelerated partition. Times normalized to the smallest replication number.

The strong scaling behaviour is similar for different replication numbers, depending on the number of processes, but differs in terms of the length of the tail, depending on the total system size and thus the amount of computational work required to compute fields and energy.

A clear parallel overhead is evident when using two or more GPUs, as the runtime shows a significant increase when using two GPUs for any given replication number. This increase in computation time can be attributed to the parallel computation of the FFT. FFTs are solved so efficiently that any necessary communication creates significant parallel overhead. Furthermore, the results show that this parallel overhead scales with regard to computation time on two GPUs.

In the conducted benchmarks, the k-space part was computationally more demanding than the short-range computation, regardless of the split between the sub-communicators used.

A more in-depth analysis can be conducted once speed-up and parallel efficiencies have been calculated. The results in Fig. 4 highlight distinct scaling regimes depending on the problem size. For the smallest systems ($8 \times 8 \times 8$ and $16 \times 16 \times 16$), the speedup remains below ideal value and quickly saturates as the number of GPUs increases. This is also reflected in the parallel efficiency, which drops sharply below 10% already at moderate GPU counts. The dominant factor here is parallel overhead: communication and synchronization costs, particularly those associated with the FFT-based long-range solver, outweigh the relatively small computational workload.

As a result, adding more GPUs does not translate into proportional performance gains, and in some cases even leads to diminishing returns. This behavior is typical for strong scaling of small problem sizes on modern GPU clusters, where latency and communication costs become the limiting factors.

In contrast, larger systems ($32 \times 32 \times 32$ and $64 \times 64 \times 64$) exhibit significantly improved scaling behavior. The speedup follows the ideal trend much more closely over a wide range of GPU counts, and the parallel efficiency remains close to unity for intermediate configurations, even exceeding it slightly in some cases likely due to cache and load-balancing effects. Only at higher GPU counts does a gradual efficiency drop become visible, indicating the onset of communication-dominated execution. These results demonstrate that the computational intensity of the

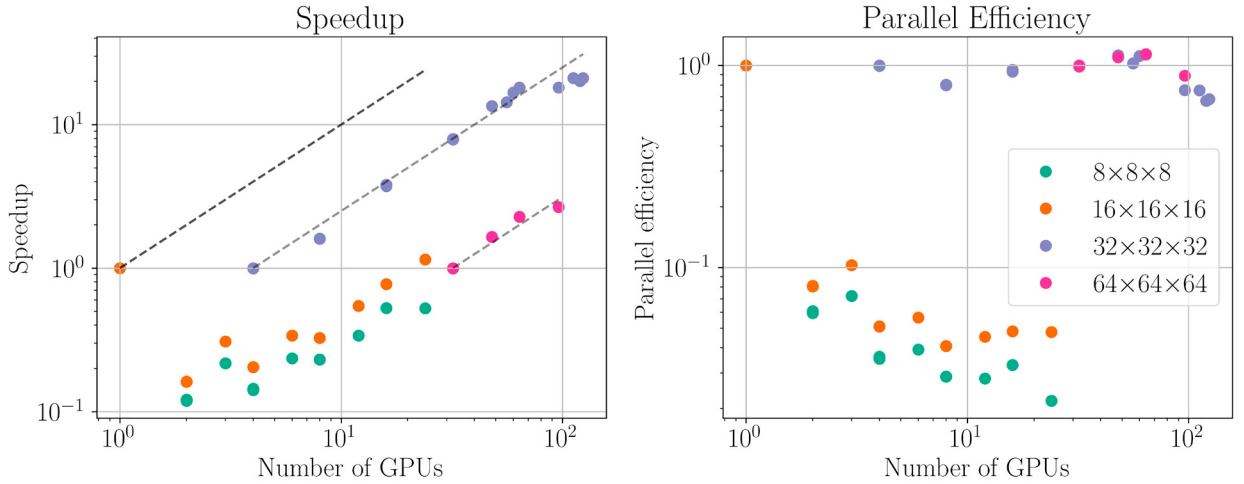


Fig. 4. Speedup (left) and Parallel efficiency (right) of strong-scaling benchmark results for different system replication numbers (8 to 64) on 1 to 128 GPUs (1 to 32 nodes) on MareNostrum5 accelerated partition. Dashed lines represent the ideal speedup taking as reference the smallest amount of resources that could compute the system size.

larger problem sizes is sufficient to amortize the communication overhead, allowing the GPU-parallel implementation to achieve high efficiency.

Overall, Fig. 4 confirms that the mesh-based electrostatics solver can be scaled up effectively on multi-GPU systems, provided that the problem size is large enough to balance computation and communication. Nevertheless, the FFT-based long-range component remains the primary scalability bottleneck.

5. Conclusions

The results show that the P3M Coulomb solver implementation presented is capable of running on the accelerated part of the MareNostrum5 system. As discussed previously, the following conclusions can be drawn regarding the sensible use of the implementation.

Firstly, if the FFT grid required to achieve the necessary solution accuracy in the fields is small enough to fit on a single GPU, it should be executed on a single GPU due to the significant parallel processing overhead created when using two or more GPUs. Secondly, if the grid does not fit on a single GPU, it can be seen that the k-space part of the solver scales with regard to the smallest number of GPUs necessary to compute the FFT. Therefore, if the FFT part does not fit on a single GPU, it may be sensible to use more GPUs, since depending on the grid size, the k-space part benefits from the scaling.

Further analysis is needed to better define the parameters indicating when large-scale execution is sensible and when smaller-scale execution is preferable. This is because there may be a crossover point with regard to parallel overhead, at which point parallel execution may be beneficial regardless of purely memory size restrictions. With this in mind, for system sizes that fall in between the optimal scaling shown in this study for large systems, and the large parallel overhead introduced by FFT communication it might be better to explore parallel replicas, umbrella sampling or task farming depending on each user's application.

Furthermore, in this study, we limited the work assigned to the real space. Increasing the real-space splitting parameter could considerably reduce the work required to achieve the desired accuracy through k-space partitioning.

Acknowledgements

We acknowledge EuroHPC JU for awarding the project ID **EHPC-DEV-2025D11-078** access to MareNostrum5 GPP and ACC partitions hosted by Barcelona Super Computing Centre and ee thank the Jülich Supercomputing

Centre, Forschungszentrum Jülich for access to the JURECA-DC Evaluation Platform and JUWELS Booster. R. A. C. B. and G. S. acknowledge funding by the European Union; this work has received funding from the European High Performance Computing Joint Undertaking under the grant agreement [101093169](#). R. H. and G. S. acknowledge funds by the Ministry of Culture and Science (MKW) of the state of North-Rhine-Westphalia through funding of the Gauss Centre for Supercomputing (GCS).

References

- [1] M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, 2017. [doi:10.1093/oso/9780198803195.001.0001](#).
- [2] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, L. G. Pedersen, A smooth particle mesh ewald method, *The Journal of Chemical Physics* 103 (19) (1995) 8577–8593. [doi:10.1063/1.470117](#).
- [3] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Boltz, H. Dachselt, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, G. Sutmann, Comparison of scalable fast methods for long-range interactions, *Phys. Rev. E* 88 (2013) 063308. [doi:10.1103/PhysRevE.88.063308](#).
- [4] P. P. Ewald, Die berechnung optischer und elektrostatischer gitterpotentiale, *Annalen der Physik* 369 (3) (1921) 253–287. [doi:10.1002/andp.19213690304](#).
- [5] R. W. Hockney, J. W. Eastwood, *Computer simulation using particles*, crc Press, 2021.
- [6] T. Darden, D. York, L. Pedersen, Particle mesh ewald: An nlog(n) method for ewald sums in large systems, *The Journal of Chemical Physics* 98 (12) (1993) 10089–10092. [doi:10.1063/1.464397](#).
- [7] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, E. Lindahl, Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs, *The Journal of Chemical Physics* 153 (13) (2020) 134110. [doi:10.1063/5.0018516](#).
- [8] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, S. J. Plimpton, LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales, *Comp. Phys. Comm.* 271 (2022) 108171. [doi:10.1016/j.cpc.2021.108171](#).
- [9] M. Deserno, C. Holm, How to mesh up ewald sums. i. a theoretical and numerical comparison of various particle mesh routines, *The Journal of Chemical Physics* 109 (18) (1998) 7678–7693. [doi:10.1063/1.477414](#).
- [10] S. Slattery, S. T. Reeve, C. Junghans, D. Lebrun-Grandié, R. Bird, G. Chen, S. Fogerty, Y. Qiu, S. Schulz, A. Scheinberg, A. Isner, K. Chong, S. Moore, T. Germann, J. Belak, S. Mniszewski, Cabana: A performance portable library for particle-based simulations, *Journal of Open Source Software* 7 (72) (2022) 4115. [doi:10.21105/joss.04115](#).
- [11] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, J. Wilke, Kokkos 3: Programming model extensions for the exascale era, *IEEE Transactions on Parallel and Distributed Systems* 33 (4) (2022) 805–817. [doi:10.1109/TPDS.2021.3097283](#).
- [12] A. Ayala, S. Tomov, A. Haidar, J. Dongarra, hefft: Highly efficient fft for exascale, in: *Computational Science – ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part I*, Springer-Verlag, Berlin, Heidelberg, 2020, p. 262–275. [doi:10.1007/978-3-030-50371-0_19](#).
- [13] R. Halver, S. Schulz, G. Sutmann, ALL - A loadbalancing library, C++ / Fortran library, <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing/-/releases>.
URL <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing/-/releases>
- [14] R. A. Bartolomeu, R. Halver, J. H. Meinke, G. Sutmann, Effect of implementations of the n-body problem on the performance and portability across gpu vendors, *Future Generation Computer Systems* 169 (2025) 107802. [doi:10.1016/j.future.2025.107802](#).
- [15] R. Halver, C. Junghans, G. Sutmann, Kokkos-based implementation of mpcd on heterogeneous nodes, in: R. Wyrzykowski, J. Dongarra, E. Deelman, K. Karczewski (Eds.), *Parallel Processing and Applied Mathematics*, Springer International Publishing, Cham, 2023, pp. 3–13. [doi:10.1007/978-3-031-30445-3_1](#).
- [16] M. Deserno, C. Holm, How to mesh up ewald sums. ii. an accurate error estimate for the particle–particle–particle-mesh algorithm, *The Journal of Chemical Physics* 109 (18) (1998) 7694–7701. [doi:10.1063/1.477415](#).