

A Massively Parallel Performance Portable Free-space Spectral Poisson Solver

Sonali Mayani^{a,b}, Veronica Montanaro^b, Antoine Cerfon^{c,d}, Matthias Frey^e,
Sriramkrishnan Muralikrishnan^f, Andreas Adelmann^{a,b}

^a*Paul Scherrer Institute, Villigen, Switzerland*

^b*ETH Zürich, Switzerland*

^c*Courant Institute of Mathematical Sciences, New York University, USA*

^d*Type One Energy Group - Canada Inc, Vancouver, BC, Canada*

^e*Mathematical Institute, University of St Andrews, KY16 9SS, UK*

^f*Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany*

Abstract

Vico et al. (2016) suggest a fast algorithm for computing volume potentials, beneficial to fields with problems requiring the solution of Poisson’s equation with free-space boundary conditions, such as the beam and plasma physics communities. Currently, the standard method for solving the free-space Poisson equation is the algorithm of Hockney and Eastwood (1988), which is second order in convergence at best. The algorithm proposed by Vico et al. converges spectrally for sufficiently smooth functions i.e. faster than any fixed order in the number of grid points. In this paper, we implement a performance portable version of the traditional Hockney-Eastwood and the novel Vico-Greengard Poisson solver as part of the IPPL (Independent Parallel Particle Layer) library. For sufficiently smooth source functions, the Vico-Greengard algorithm achieves higher accuracy than the Hockney-Eastwood method with the same grid size, reducing the computational demands of high resolution simulations since one could use coarser grids to achieve them. More concretely, to get a relative error of 10^{-4} between the numerical and analytical solution, one requires only 16^3 grid points in the former, but 128^3 in the latter, more than a 99% memory footprint reduction. Additionally, we propose an algorithmic improvement to the Vico-Greengard method which further reduces its memory footprint. This is particularly important for GPUs which have limited memory resources, and should be taken into account when selecting numerical algorithms for performance portable codes. Finally, we showcase performance through GPU and CPU scaling studies on the Perlmutter (NERSC) supercomputer, with efficiencies staying above 50% in the strong scaling case.

Keywords: Fast free-space Poisson solvers, Exascale, Performance portability, PIC

1. Introduction

Simulation codes in many areas of physics need to solve elliptic partial differential equations (PDEs), such as the Poisson equation or the Helmholtz equation. Many computational methods have been developed for this purpose. In particular, we look at integral-based approaches for open boundary conditions, as is customary in previous work [1, 2, 3]. This approach requires dealing with free-space Green’s functions, which are singular at the origin and long-range in nature, requiring special attention during computational implementations.

We take the example of the Poisson equation in the context of electrostatics. The PDE governing our system in this case is given by:

$$\Delta\phi(\vec{x}) = -\frac{\rho(\vec{x})}{\epsilon_0}, \quad (1)$$

where $\rho(\vec{x})$ is the charge density and $\phi(\vec{x})$ the scalar potential, ϵ_0 the vacuum permittivity, and $\vec{x} \in \mathbb{R}^3$. For free-space boundaries ($\phi(\vec{x}) \rightarrow 0$ as $|\vec{x}| \rightarrow \infty$), the solution ϕ can be written as a convolution:

$$\phi(\vec{x}) = \frac{1}{\epsilon_0} \int G(\vec{x} - \vec{x}') \rho(\vec{x}') d\vec{x}', \quad (2)$$

where the Green’s function $G(\vec{x}) = 1/(4\pi|\vec{x}|)$ is the solution of $\Delta\phi = -\delta(\vec{x})$. A computational brute-force approach would consist of discretizing G and ρ on the computational mesh, and approximating the integral as a sum. However, the computational cost of such an operation is $\mathcal{O}(N^2)$ [4], where $N = N_x N_y N_z$, with N_x , N_y , and N_z being the number of grid points in each direction of the 3D grid.

For uniform meshes, the computational cost can be reduced by making use of fast Fourier transforms (FFTs). The convolution theorem states that in Fourier space a convolution can be written as a multiplication [5]. Equation (2) can therefore be rewritten as $\phi = \mathcal{F}^{-1}\{\mathcal{F}\{G\}\mathcal{F}\{\rho\}\}$ where $\mathcal{F}$ denotes a Fourier transform and $\mathcal{F}^{-1}$ its inverse. This approach drastically reduces the cost to $\mathcal{O}(N\log N)$. However, FFTs work for periodic signals, and in this case ρ and G are not periodic, since we are interested in free space boundary conditions. Hockney and Eastwood introduced an algorithm which converts the convolution to a cyclic one, making all involved signals periodic, while still obtaining the correct free-space solution at the end [1].

Vico, Greengard, and Ferrando have proposed a novel, fast and spectrally accurate method which can be applied to many scientific computing problems requiring a convolution with a free-space Green’s function [6]. For smooth distributions, the accuracy of this solver converges faster than any fixed polynomial order of the gridsize as the resolution is increased, whereas the Hockney-Eastwood algorithm is only second-order accurate. As suggested in [7], this method could replace the Hockney-Eastwood algorithm for beam and plasma physics problems, and provide better accuracy with lower resolution and therefore a smaller memory footprint. Using a faster, more accurate, and less memory intensive code is suitable for memory-bound architectures such as GPUs,

which are becoming increasingly common in scientific computing and super-computing clusters.

However, the new Vico-Greengard method adds another memory bottleneck, even if there is a gain in accuracy: while in the standard Hockney-Eastwood method one needs to double the domain $(2N)^3$, in the new method a pre-computation of the Green’s function on a $(4N)^3$ grid is required to capture the full frequency content [6]. Storing this $(4N)^3$ grid causes the memory footprint of the Poisson solver to double for each dimension, resulting in an eight-fold memory increase in total.

We propose an algorithmic improvement by changing from a discrete Fourier transform to the discrete cosine transform to constrain the pre-computation to a $(2N + 1)^3$ grid, reducing the memory footprint to be similar to the Hockney-Eastwood algorithm. With this improvement, the novel Vico-Greengard method becomes attractive both from the memory requirement as well as its ability to deliver high accuracy solutions.

In this paper, we present a parallel, performant, and portable version of the algorithm originally proposed by Vico, Greengard, and Ferrando [6]. We outline our implementation in Section 2. A comprehensive evaluation of our implementation through convergence studies, scaling studies, and memory benchmarks, demonstrating its superior performance compared to the state-of-the-art Hockney-Eastwood algorithm is given in Section 3.

2. Methodology

2.1. IPPL: Independent Parallel Particle Layer

IPPL [8] is an open-source C++ framework that provides the tools to develop particle-mesh methods, such as particle-in-cell, explained in Appendix A.1. The framework is built upon Kokkos [9] which enables cross-platform performance portability for shared memory parallelism. Inter-process communication takes place via the message passing interface (MPI), the de facto standard for distributed memory parallelism. IPPL further depends on the heFFTe (highly efficient FFT for exascale) library to carry out fast Fourier transforms (FFTs). A general overview of the dependencies and features of IPPL is depicted in Figure 1. IPPL has a set of benchmark problems in plasma physics applications, namely ALPINE [10], which helps in developing novel algorithms, optimizing HPC implementations, and benchmarking individual components. Further details about the structure of IPPL can be found in [8, 10].

It is in the context of these particle-mesh methods in IPPL that we seek to solve the Poisson equation (Eq. 1). As mentioned previously, for free-space boundary conditions, this is usually done by writing the solution in terms of a convolution, which is efficiently computed in Fourier space and then Fourier transformed back to real space. Until recently, the standard method for doing so was the Hockney-Eastwood method, explained in the following section.

2.2. Hockney-Eastwood Method

In order to make the calculation of the convolution in Eq. (2) efficient, the Hockney-Eastwood method makes the convolution cyclic, such that FFTs can be used. Assuming

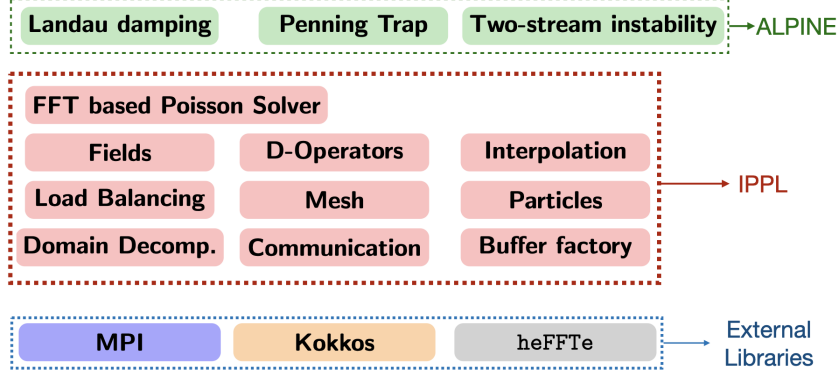


Figure 1: A general overview of the IPPL framework and its external library dependencies. The framework provides the essential components to develop particle-mesh methods. A set of mini-applications is gathered in ALPINE [10] which is currently distributed as part of IPPL.

a 3-dimensional physical domain of size $[0, L_x] \times [0, L_y] \times [0, L_z]$ and a mesh of $N_x \times N_y \times N_z$ grid-points, such that the mesh spacing is $h_x = L_x/N_x$, $h_y = L_y/N_y$, $h_z = L_z/N_z$ in the corresponding directions, the steps of the algorithm are as follows [1]:

1. Double the computational grid in the non-periodic directions (those which have open boundary conditions). If all boundary conditions are open, this means we will have a grid of size $2N_x \times 2N_y \times 2N_z$.
2. The source term $\rho(\vec{x})$, which is defined on $\vec{x} \in [0, L_x] \times [0, L_y] \times [0, L_z]$, is then zero-padded to be defined on the full doubled domain. So for $\vec{x} \in [0, 2L_x) \times [0, 2L_y) \times [0, 2L_z)$, we define $\rho_2(\vec{x})$ such that:

$$\rho_2(\vec{x}) = \begin{cases} \rho(\vec{x}), & \text{if } \vec{x} \in [0, L_x) \times [0, L_y) \times [0, L_z) \\ 0, & \text{otherwise.} \end{cases}$$

3. The Green's function $G(\vec{x})$ also needs to be extended to the doubled domain. This is accomplished by circular shifting and making $G(\vec{x})$ periodic, such that the new function $G_2(\vec{x})$ satisfies:

$$G_2(\vec{x}) = \begin{cases} G(\vec{x}), & \text{if } \vec{x} \in [0, L_x) \times [0, L_y) \times [0, L_z) \\ G(2L_x - x, y, z), & \text{if } \vec{x} \in [L_x, 2L_x) \times [0, L_y) \times [0, L_z) \\ G(x, 2L_y - y, z), & \text{if } \vec{x} \in [0, L_x) \times [L_y, 2L_y) \times [0, L_z) \\ G(x, y, 2L_z - z), & \text{if } \vec{x} \in [0, L_x) \times [0, L_y) \times [L_z, 2L_z) \\ G(2L_x - x, 2L_y - y, z), & \text{if } \vec{x} \in [L_x, 2L_x) \times [L_y, 2L_y) \times [0, L_z) \\ G(2L_x - x, y, 2L_z - z), & \text{if } \vec{x} \in [L_x, 2L_x) \times [0, L_y) \times [L_z, 2L_z) \\ G(x, 2L_y - y, 2L_z - z), & \text{if } \vec{x} \in [0, L_x) \times [L_y, 2L_y) \times [L_z, 2L_z) \\ G(2L_x - x, 2L_y - y, 2L_z - z), & \text{otherwise.} \end{cases}$$

The Green's function is singular at the origin and must therefore be regularized. It is customary to replace the singularity by $G(\vec{0}) = -1/4\pi$.

4. Since ρ_2 and G_2 are now periodic with period $(2L_x, 2L_y, 2L_z)$, we can Fourier transform them (after discretization) using FFTs. In the Fourier domain, the convolution becomes a simple multiplication of the Fourier transforms of ρ_2 and G_2 . In this case, the potential on the doubled domain is given by ϕ_2 :

$$\phi_2 = h_x h_y h_z \mathcal{F}^{-1} \{ \mathcal{F} \{ \rho_2 \} \times \mathcal{F} \{ G_2 \} \}.$$

5. The physical solution for the potential is obtained by restricting the double-grid solution to the physical domain, i.e. $\phi(\vec{x}) = \phi_2(\vec{x})$ for $\vec{x} \in [0, L_x] \times [0, L_y] \times [0, L_z]$. The proof of the equivalence in the physical domain between the cyclic convolution and the original convolution can be found in [11, p. 12].

The computational cost to solve the Poisson equation with this method is $\mathcal{O}((8N)\log(8N))$, where $N = N_x N_y N_z$. The trade-off is the increased memory requirement to store the fields ρ_2 and G_2 on the doubled domain. The Hockney-Eastwood method is second-order accurate. Thus, if high accuracy is desired, the computation quickly becomes expensive. It is therefore advantageous to explore more accurate methods, such as the spectrally accurate Vico-Greengard method [7], which is briefly summarised below.

2.3. Vico-Greengard Method

Vico, Greengard, and Ferrando describe a new method which differs from Section 2.2 by the choice of Green's function. However, this change enables spectral accuracy for smooth functions [6]. For the Poisson equation, where the differential operator is the Laplace operator, the choice of the Green's function in the Fourier space as per [6] is

$$G_L(\vec{s}) = 2 \left(\frac{\sin(L|\vec{s}|/2)}{|\vec{s}|} \right)^2, \quad (3)$$

where $\vec{s} \in \mathbb{R}^3$ is in frequency domain and L is the truncation window size, which must be chosen larger than the maximum distance between any two points in the computational domain [6], i.e. $L = \alpha \sqrt{L_x^2 + L_y^2 + L_z^2}$ with $\alpha > 1$ for a domain $[0, L_x] \times [0, L_y] \times [0, L_z]$. Here, we have chosen $\alpha = 1.1$.

To avoid any loss of information when calculating the inverse Fourier transform of Eq. (3), the transform must be performed on a $4N_x \times 4N_y \times 4N_z$ grid due to the oscillatory nature of G_L . First, as explained by Hockney and Eastwood [1, p. 213]), we need to have a grid of double the original size to perform the aperiodic convolution. The additional factor of two in the mesh size compared to the Hockney-Eastwood method comes from the fact that G_L is an oscillatory signal. As stated by the Nyquist sampling theorem, an oscillatory signal must be sampled at a sampling rate f_s two times larger than the maximum frequency $f_{\max}$ of that signal, i.e. $f_s > 2f_{\max}$, to not lose any information.

However, if all the convolution computation is performed on the quadruple grid, the source term must also be zero-padded on the quadruple domain, which makes the

method prohibitively memory-intensive for computational simulations. Fortunately, this can be avoided by calculating the inverse transform of G_L on the fourfold grid in only one pre-computation step, which is then restricted to the doubled grid of size $2N_x \times 2N_y \times 2N_z$ to obtain G_2 . The restricted Green's function G_2 is afterwards reused every time step in which the Poisson solve is called during the simulation. The quadruple grid is therefore never needed again during the course of the simulation unless the grid spacing changes in which case the Green's function again needs to be updated. Thus, the algorithm follows the same steps as in Section 2.2 except for the Green's function computation, where we use the pre-computed G_2 . In this case, the complexity is similar to the Hockney-Eastwood method, but the method has a higher accuracy for sufficiently smooth functions. The main drawback is the need for a $4N_x \times 4N_y \times 4N_z$ grid in the pre-computation step, which increases memory demands by a factor of eight compared to the Hockney-Eastwood method. This is especially a problem with GPU architectures, which generally have less memory available than their CPU counterparts. In order to circumvent this, we present an algorithmic change to reduce the memory required by the pre-computation step.

2.4. A Modified Vico-Greengard Method

A simple modification in the above algorithm allows us to reduce the grid size of the pre-computation step from $(4N_x \times 4N_y \times 4N_z)$ to $((2N_x + 1) \times (2N_y + 1) \times (2N_z + 1))$. We make use of the fact that the Green's function G_L used in the Vico-Greengard method, given by Eq. (3), is purely real and has even symmetry i.e. $G_L(\vec{s}) = G_L(-\vec{s})$. In view of the even symmetry, the Green's function on the Fourier domain G_L , which is defined on a $4N_x \times 4N_y \times 4N_z$ grid to contain the full frequency content, actually only has $((2N_x + 1) \times (2N_y + 1) \times (2N_z + 1))$ unique values. Furthermore, the discrete Fourier transform reduces to a discrete cosine transform (DCT) thanks to the same properties of the signal, namely that G_L is purely real and even. In this way, we can perform the pre-computation step of the Vico-Greengard method on the $(2N + 1)^3$ domain instead of the fourfold domain, and use the DCT to compute the inverse transform of G_L . This way, the memory footprint of the Vico-Greengard method is the same order-of-magnitude to the Hockney-Eastwood method.

3. Results and Discussion

3.1. Code verification and accuracy analysis

To test the correctness and accuracy of the numerical methods used in our implementation, we use the analytical solution of the Poisson's equation for a Gaussian source

$$\rho = -\frac{1}{\sqrt{(2\pi)^3}} \exp\left(-\frac{r^2}{2\sigma^2}\right), \quad (4)$$

which is given by [7]

$$\phi_{exact} = \frac{1}{4\pi r} \operatorname{erf}\left(\frac{r}{\sqrt{2}\sigma}\right),$$

with $r = \sqrt{(x - \mu)^2 + (y - \mu)^2 + (z - \mu)^2}$. We position the Gaussian in the center of a unit box, i.e. $\mu = 0.5$ and $L_x = L_y = L_z = 1$, with standard deviation $\sigma = 0.05$.

In a second test case, we show the behaviour of the numerical schemes when the source is non-smooth. The test case described in [12] has a source term given by

$$\rho = \begin{cases} 4\pi G, & \text{if } r \leq 1 \\ 0, & \text{if } r > 1 \end{cases}, \quad (5)$$

and the exact solution

$$\phi_{exact} = \begin{cases} -\frac{2}{3}\pi G(3 - r^2), & \text{if } r \leq 1 \\ -\frac{4}{3}\pi \frac{G}{r}, & \text{if } r > 1 \end{cases},$$

where $G = 6.67408 \cdot 10^{-11} \text{ m}^3\text{kg}^{-1}\text{s}^{-2}$ is the gravitational constant and r is as above. We choose $\mu = 1.2$ to center the distribution in the 3D cubic box of extents $L_x = L_y = L_z = 2.4$. The solution ϕ is the gravitational potential of a sphere.

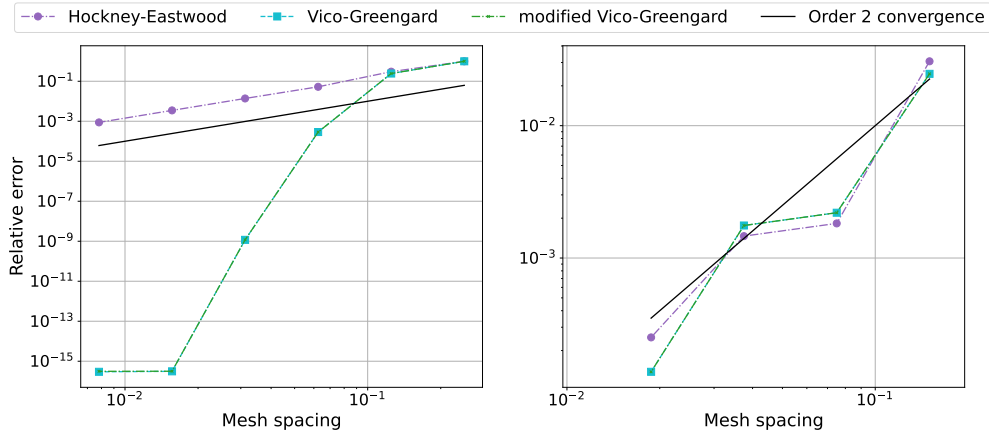


Figure 2: Relative error between analytical and computed solutions for all three solvers (Hockney-Eastwood, Vico-Greengard, and its modified version). The 2nd-order convergence theoretical line is plotted for reference. Left: Source term given by Equation 4. Right: Source term given by Equation 5.

In order to compare the numerical and analytical solutions, we use the relative error L2 norm. By means of a convergence study we compare the error convergence rates with their theoretical values to verify the correctness of the implementation. In Figure 2, we obtain the expected second-order convergence with the Hockney-Eastwood solver, whereas with the Vico-Greengard solver we get spectral convergence for the smooth source and second-order convergence for the non-smooth source terms. The modified Vico-Greengard solver follows the same trend as the original, therefore proving the correctness of our algorithmic modifications.

These results also shed light on the limitations of the new solver: the convergence of Vico-Greengard solver depends on the smoothness of the source term. For the second test case with a non-smooth source term, i.e. Equation 5, the Vico-Greengard solver

is only as good as the Hockney-Eastwood solver. However, since the modified Vico-Greengard solver does not cause any additional computational costs in terms of memory and complexity compared to the Hockney-Eastwood solver, it can replace it as the default method. The accuracy is therefore limited by the properties of the source term instead of the solver itself.

3.2. Scaling studies

We run strong and weak scaling studies of both the Hockney-Eastwood and Vico-Greengard solvers implemented in IPPL. These studies are conducted on the Perlmutter system, a HPE Cray machine at NERSC (USA). To showcase portability and scaling across architectures, the studies are done both on the GPU and CPU partitions. The GPU partition contains 1536 nodes where each of them is equipped with 4 NVIDIA A100 GPUs. The CPU partition consists of 3072 nodes, with 2 AMD EPYC 7763 CPUs of 64 cores each. We run the simulations with 1 MPI rank per GPU for the GPU scaling studies and 128 MPI ranks per CPU node (1 MPI task per physical core) for the CPU scaling studies.

The strong scaling study is done by keeping the problem size fixed and measuring the timing as we increase the number of nodes for the simulation. We run this for two different problem sizes: 512^3 and 1024^3 . The weak scaling study consists in increasing the number of nodes used for the simulation while maintaining a constant workload, i.e. we increase the problem size with the number of nodes. Starting with 512^3 as our problem size, we increase to $512^2 \times 1024$ when doubling the number of nodes, then 512×1024^2 for four times the initial node count and finally ending at 1024^3 .

IPPL was compiled with Cuda 12.0 (GPU-specific), FFTW 3.3.10.5 (CPU-specific), Kokkos 4.1.0, Heffte 2.4.0, and manual installations of MPICH (due to some technical issues, a manual installation of MPICH was needed in order to be able to compile with Cuda 12 on the cluster). One small caveat is that the MPICH version used to compile IPPL on CPU is 4.1.2 whereas for GPU it is 4.1.1, due to some technical issues on the cluster. Since this is a minor version change with bug fixes related to “user-reported crashes and build issues” [13], we do not expect differences in our scaling studies.

In Fig. 3 we show the GPU strong scaling results of the 512^3 grid simulation for both the Hockney-Eastwood algorithm (left panel) and the modified Vico algorithm (right panel). The blue lines denote the overall algorithm execution times. The timings of the forward and backward FFT transforms are highlighted in green. The costs for the data transfer between the grids of size N^3 and $(2N)^3$ and vice versa are shown in red. The algorithms have nearly identical execution times and they show very good scaling behaviour up to 256 nodes (i.e. 1024 GPUs). The parallel efficiency does not drop below 65%. After 32 nodes, the data copy kernel timings (red lines) start to flatten since the communication costs associated with the data transfer break even with the actual data copy. The CPU strong scaling of the same setup is given in Fig. 4. We again observe very good scaling behaviour, i.e. the parallel efficiency remains above 50%. Note that here we start with 1 node rather than 4 nodes as for the GPU scaling due to the smaller amount of memory on the GPUs. Also, the data transfer timings do not flatten as we

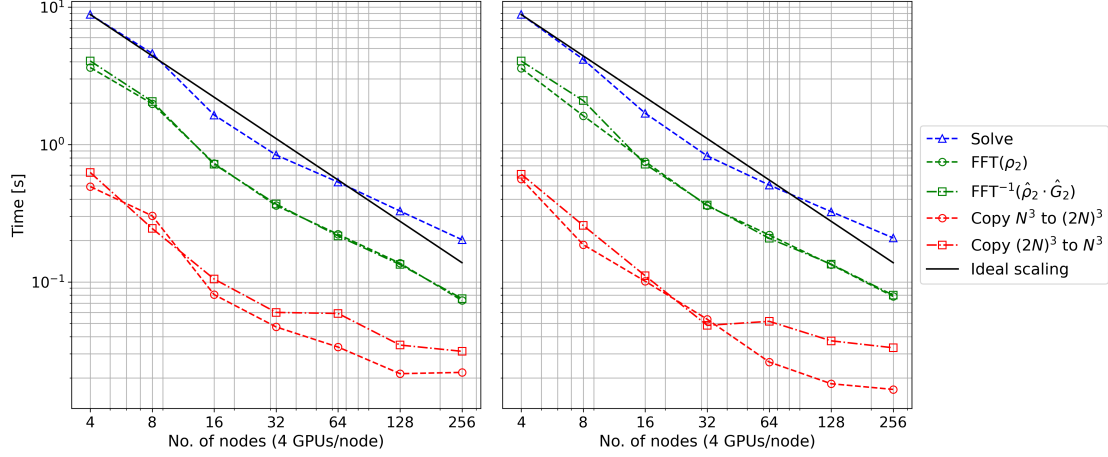


Figure 3: Strong scaling results on GPU for both the Hockney-Eastwood (left) and the modified Vico-Greengard solver (right) with a problem size of $N^3 = 512^3$, using the `heFFTe` parameters of pencil decomposition, a2av communication, no reordering, and with GPU-aware enabled. The efficiency stays above 65%.

have seen for the GPU runs because the actual data copy on the CPU is slower than on the GPU and therefore hides the associated communication costs. The dominating cost on both hardware systems (GPU and CPU) are the FFT transforms as expected. Note that the FFT timers also include communication costs.

The GPU and CPU strong scalings for a problem size of 1024^3 grid points are provided in Fig. 5 and Fig. 6, respectively. We obtain similar scaling behaviour on both hardware systems as for 512^3 grid points and the parallel efficiency remains above 65%.

The weak scaling results on both hardware systems are given in Fig. 7 and Fig. 8. We start at a problem size of 512^3 grid points and increase the problem size up to 1024^3 grid point while keeping the workload per node approximately constant. On both hardware systems we experience a loss in parallel efficiency because the cost of the all-to-all communication in the FFTs increases with the node count while the workload is kept constant.

3.3. Memory footprint

We can quantify the memory gain obtained from using the discrete cosine transform by measuring the memory usage on one processor for all three solvers: the Hockney-Eastwood solver, the Vico-Greengard solver, and the modified version of Vico-Greengard. The CPU results are shown in Figure 9, and are similar for the GPU case. As we can see, without the improvement, the memory footprint of Vico-Greengard is ~ 5 -8 times larger than the Hockney-Eastwood method. With the algorithmic modification of using the discrete cosine transform, the memory footprint is brought down to be similar to the Hockney-Eastwood method. The Vico-Greengard algorithm therefore becomes a viable solver for the free-space Poisson equation in the context of mesh-based methods.

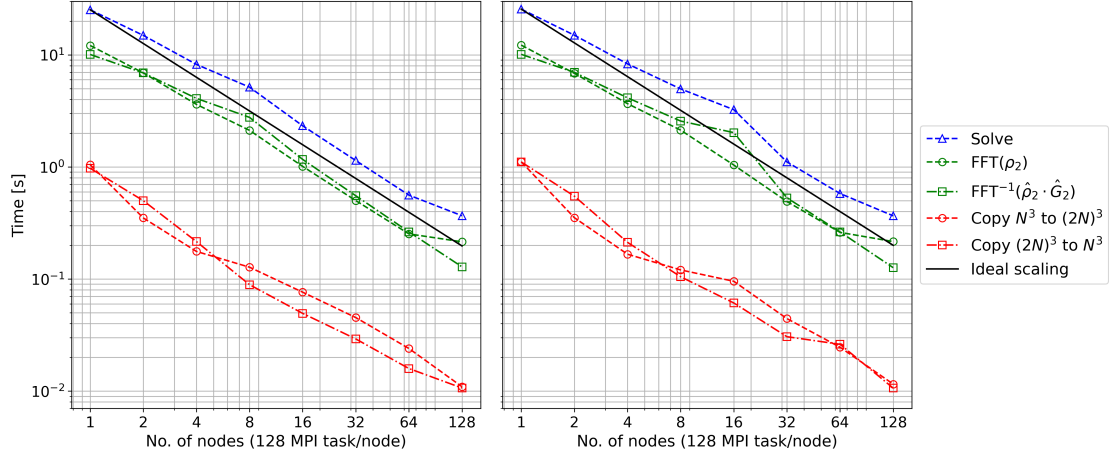


Figure 4: Strong scaling results on CPU for both the Hockney-Eastwood (left) and the modified Vico-Greengard solver (right) with a problem size of $N^3 = 512^3$, using the **heFFTe** parameters of pencil decomposition, a2av communication, no reordering. The efficiency stays above 50%.

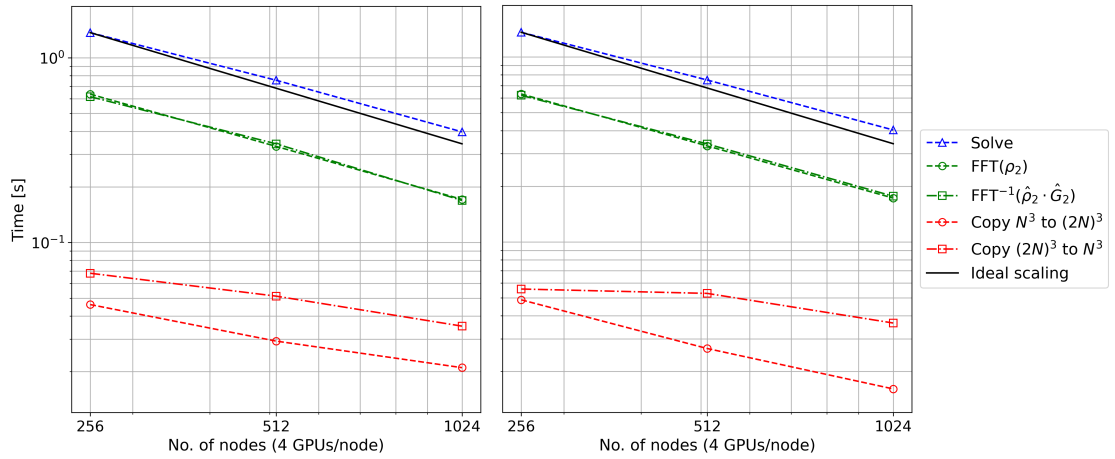


Figure 5: Strong scaling results on GPU for both the Hockney (left) and the modified Vico solver (right) with a problem size of $N^3 = 1024^3$, using the **heFFTe** parameters of pencil decomposition, a2av communication, no reordering, and with GPU-aware enabled. The efficiency stays above 80%.

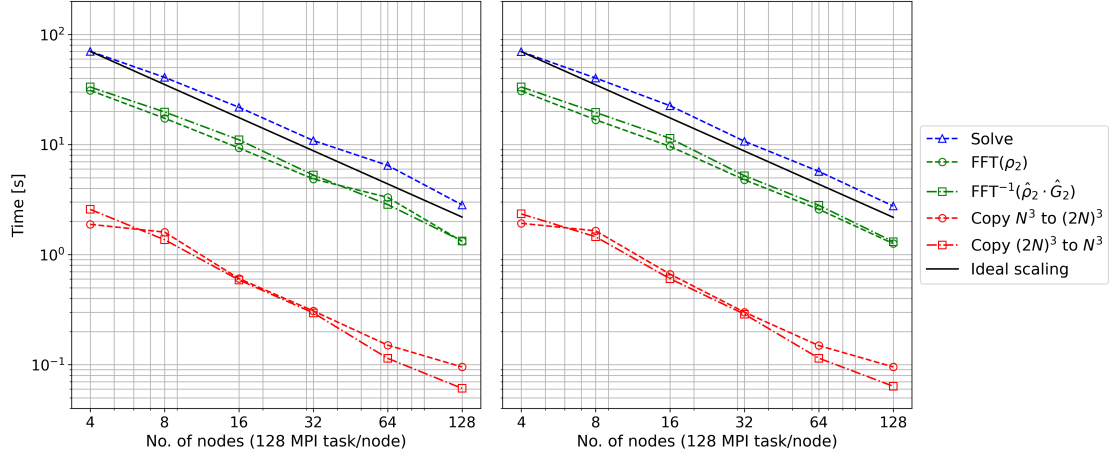


Figure 6: Strong scaling results on CPU for both the Hockney-Eastwood (left) and the modified Vico-Greengard solver (right) with a problem size of $N^3 = 1024^3$, using the **heFFTe** parameters of pencil decomposition, a2av communication, no reordering. The efficiency stays above 75%.

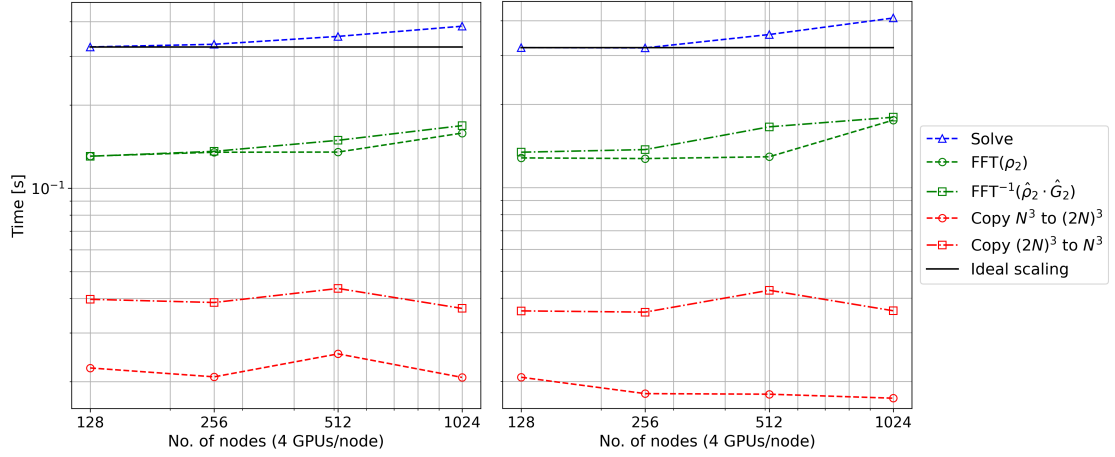


Figure 7: Weak scaling results on GPU for both the Hockney-Eastwood (left) and the modified Vico-Greengard solver (right), starting from a problem size of $N^3 = 512^3$ and increasing the workload proportionally to the node increase until reaching a problem size of $N^3 = 1024^3$. The **heFFTe** parameters used are pencil decomposition, a2av communication, no reordering, and GPU-aware enabled. The efficiency stays above 75%.

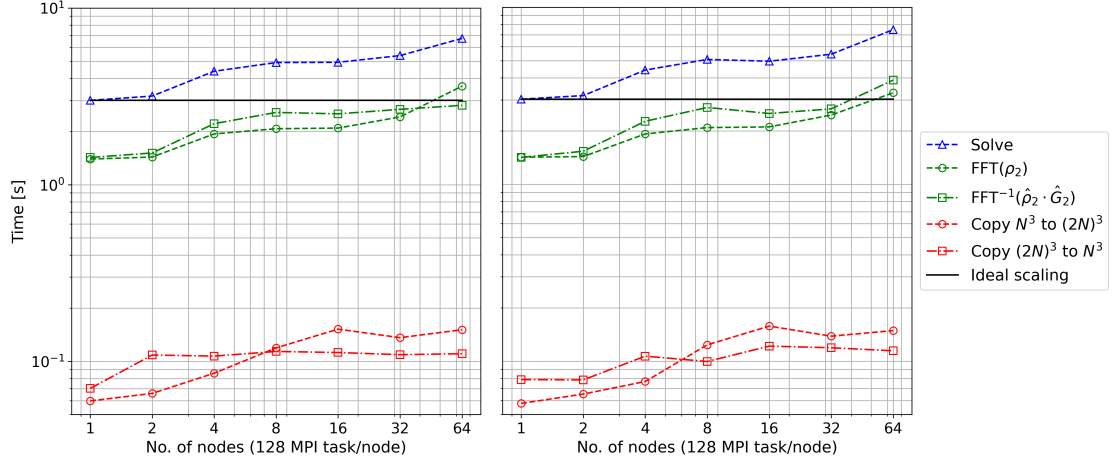


Figure 8: Weak scaling results on CPU for both the Hockney-Eastwood (left) and the modified Vico-Greengard solver (right), starting from a problem size of $N^3 = 512^3$ and increasing the workload proportionally to the increase in node counts until reaching a problem size of $N^3 = 1024^3$. The **heFFTe** parameters used are pencil decomposition, a2av communication, no reordering. The efficiency goes down to 40%.

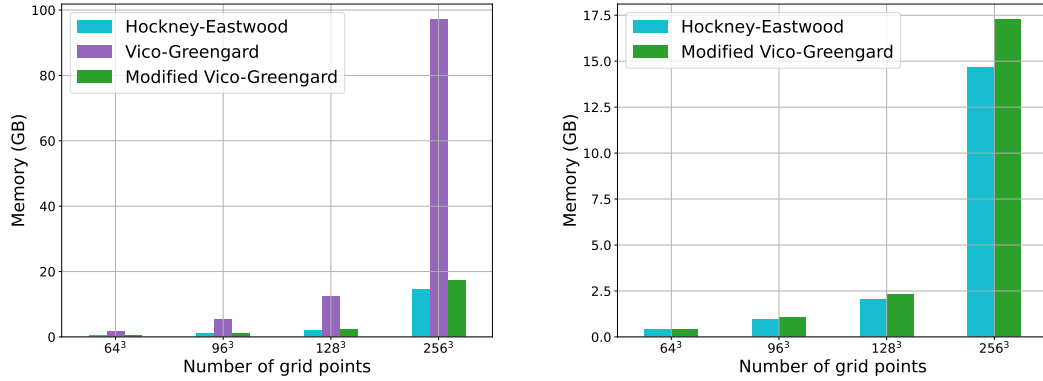


Figure 9: Memory footprints for various problem sizes obtained using Kokkos Highwater on CPU. Left: Memory comparison of Hockney-Eastwood, original Vico-Greengard, and modified Vico-Greengard solvers. Right: Comparison of only Hockney-Eastwood and the modified Vico-Greengard solver, which has an algorithmic optimization.

4. Conclusion

In this work, we introduce an enhancement to the fast free space Poisson solver initially proposed by Vico et al. in [6]. Our improvement reduces the memory usage of the algorithm to just one-eighth of its original amount, bringing its memory footprint into the same order-of-magnitude range as the Hockney-Eastwood solver [1], the current state-of-the-art method for addressing the free space Poisson problem in particle-mesh simulations. The new solver offers a significant advantage by providing spectral accuracy for smooth source terms. As a result, it needs fewer grid points compared to the state-of-the-art second-order Hockney-Eastwood solver to achieve the same level of accuracy for sufficiently smooth distributions. This makes it an excellent choice for overcoming memory and runtime constraints in large-scale, high-resolution simulations.

We have implemented this newly improved solver in a massively parallel and portable framework targeted to exascale machines. We quantify the correctness and accuracy improvements with respect to the Hockney-Eastwood method, as well as the memory improvements with respect to the original algorithm in [6]. Finally, we demonstrate the efficiency of the solver by running strong and weak scaling analyses on Perlmutter, for both CPU and GPU, using up to $\sim 70\%$ of the full machine in the latter case. The results of the strong scaling study show that the efficiency stays above 50% for the problem size of 512^3 , and above 65% in the case of the larger problem size 1024^3 .

Currently, the new solver has been verified in the context of a plasma physics application, a charge-neutral Penning trap, the results of which can be found in Appendix A.

In future work, we will apply this free space solver in the context of beam dynamics simulations using OPAL [2], a particle accelerator modelling code, which is currently being interfaced with the IPPL framework.

Availability

IPPL is an open source project. The source code can be found here: <https://github.com/IPPL-framework/ippl>. The version of IPPL used for this study is tagged `scaling_study_vico_paper`. The versions of the build dependencies used on the Perlmutter system at NERSC are listed in the Results section of the paper. The convergence study can be found in `test/solver/TestGaussian_convergence.cpp`, and the scaling studies were done with `test/solver/TestGaussian.cpp`. The solver type can be chosen by passing either `HOCKNEY`, `VICO`, or `VICO_2`, which correspond to the Hockney-Eastwood, Vico-Greengard, and modified Vico-Greengard methods respectively.

Acknowledgements

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy Office of Science User Facility using NERSC award ERCAP-M888. The Gwendolen cluster at PSI was essential to the successful

development of the presented method. We particularly acknowledge the invaluable technical advice provided by the PSI High-Performance Computing and Engineering (HPCE) group, especially Marc Caubet Serrabou. We would also like to acknowledge Stanimire Tomov and Hartwig Anzt from the Innovative Computing Laboratory - University of Tennessee and Miroslav Stoyanov from Oak Ridge National Laboratory for their help with introducing the Discrete Cosine Transform of Type 1 in heFFTe.

Appendix A. Penning trap simulation using the Particle-In-Cell scheme

As an application of the Vico-Greengard solver, we use it in combination with the Particle-In-Cell (PIC) scheme for an electrostatic plasma physics simulation. More concretely, we simulate the dynamics of an electron bunch in a Penning trap with external electric and magnetic fields to confine them. First, we introduce the theory behind the PIC method, and then we follow-up with the results, where we compare the simulation using the Vico-Greengard solver with the Hockney-Eastwood solver.

Appendix A.1. The Particle-In-Cell (PIC) scheme

The particle-in-cell method combines particle tracking in phase space with a mesh-based approach to speedup force calculations. To reduce the computational cost needed to model physical phenomena, a cloud of plasma particles is replaced by a macro-particle with the same charge-to-mass ratio. In the case considered here, these macro-particles evolve in phase space over time according to the Vlasov-Poisson equation and Newton's second law of motion. In order to compute the force fields, the charge density of the particles is computed on the grid points of a fixed mesh (**scatter**), on which we solve for the electrostatic fields (**field solve**). These are then interpolated to the macro-particle locations (**gather**) in order to compute the force which will drive the particles to their new position at the next time-step (**particle push**) [14]. The PIC loop consists of the scatter, solve, gather, and push, and is repeated at each time-step until the end of the simulation, schematically shown in Figure A.10.

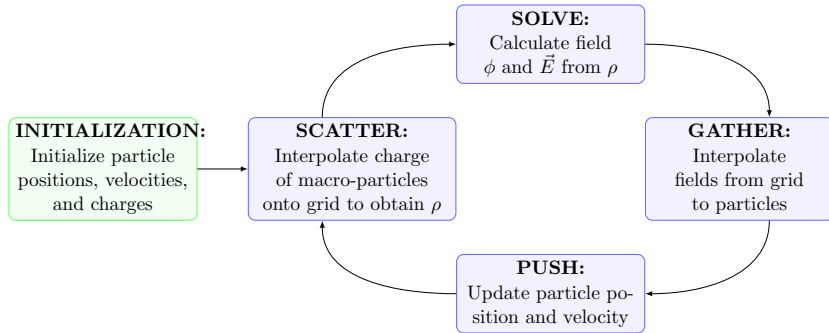


Figure A.10: The PIC loop, which is repeated every time-step after the initialization.

Let $\vec{r}_i$ be the position of macro-particle i with mass m_i and charge q_i , $\vec{p}_i = m\gamma\vec{v}_i$ its relativistic momentum, $\vec{v}_i$ its velocity, and $\gamma_i = 1/\sqrt{1 - |\vec{v}_i|^2/c^2}$ the Lorentz factor,

where c is the speed of light. The equations of motion are given by the Newton-Lorentz equations:

$$\begin{aligned}\frac{d\vec{r}_i}{dt} &= \frac{\vec{p}_i}{m_i\gamma_i} \\ \frac{d\vec{p}_i}{dt} &= q_i(\vec{E}_i + \frac{\vec{p}_i}{m_i\gamma_i} \times \vec{B}_i).\end{aligned}$$

The above are integrated using a Leapfrog algorithm [15] in the particle push phase to evolve all of the macro-particles ($i = 1, \dots, N_p$ where N_p is the number of macro-particles) to the next timestep. The fields $\vec{E}_i$ and $\vec{B}_i$ are the sum of the self-fields of the particles and any external fields that may be applied, $\vec{E}_{ext}$ and $\vec{B}_{ext}$. In the electrostatic case, the magnetic self-field is zero in the non-relativistic regime. In the case of relativistic particles, a magnetic self-field appears due to the moving charges.

Let $\vec{x} \in \mathbb{R}$ represent a position in the computational mesh, for fields (not the same as the macro-particle positions). At each time-step, the electric field $\vec{E}(\vec{x})$ is obtained from

$$\vec{E}(\vec{x}) = -\vec{\nabla}\phi(\vec{x}), \quad (\text{A.1})$$

where the potential $\phi(\vec{x})$ is computed from the charge density $\rho(\vec{x})$ using Eq. (1). After calculating the electric field $\vec{E}(\vec{x})$ on the mesh, it is interpolated as $\vec{E}_i$ to each macro-particle located at $\vec{r}_i$ (gather phase).

As mentioned in the previous section, the Poisson equation in the **solve** phase of the PIC loop is written as a convolution and solved in Fourier space. In the case of free-space boundaries, one needs to use the Hockney-Eastwood or the Vico-Greengard method as explained before.

Appendix A.2. Penning trap as a physics example

For the Penning trap, we follow the same set-up as in [10], with open boundary conditions. We do a qualitative comparison between using the Hockney-Eastwood solver and the Vico-Greengard solver by looking at the evolution of the electron charge density, as shown in Figure A.11. Both solvers produce the same expected behaviour with similar runtimes.

Identifying the regimes or test cases where the high accuracy of the Vico-Greengard solver is beneficial is beyond the scope of the current work and will be carried as a part of future work.

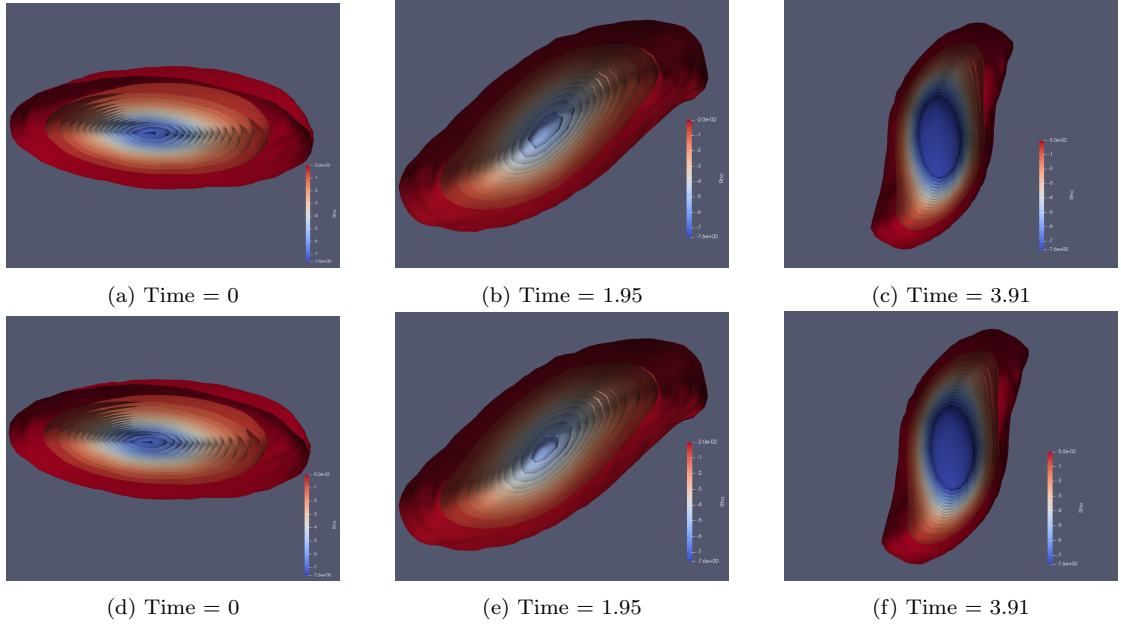


Figure A.11: Evolution of electron bunch dynamics in a 3D charge-neutral Penning trap, shown as snapshots in time during the simulation. The first and second rows correspond to using the Hockney-Eastwood and Vico-Greengard solvers, respectively.

References

- [1] R. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. CRC Press, 1988.
- [2] A. Adelman, P. Calvo, M. Frey, A. Gsell, U. Locans, C. Metzger-Kraus, N. Neveu, C. Rogers, S. Russell, S. Sheehy, J. Snuverink, and D. Winklehner, “OPAL a Versatile Tool for Charged Particle Accelerator Simulations,” *arXiv:1905.06654 [physics]*, May 2019.
- [3] R. A. James, “The solution of Poisson’s equation for isolated source distributions,” *Journal of Computational Physics*, vol. 25, pp. 71–93, Oct. 1977.
- [4] J. Qiang, S. Lidia, R. D. Ryne, and C. Limborg-Deprey, “Three-dimensional quasistatic model for high brightness beam dynamics simulation,” *Physical Review Accelerators and Beams*, vol. 9, p. 044204, Apr. 2006.
- [5] E. W. Weisstein, “Convolution Theorem.” Publisher: Wolfram Research, Inc.
- [6] F. Vico, L. Greengard, and M. Ferrando, “Fast convolution with free-space Green’s functions,” *Journal of Computational Physics*, vol. 323, pp. 191–203, Oct. 2016.
- [7] J. Zou, E. Kim, and A. J. Cerfon, “FFT-based free space Poisson solvers: why Vico-Greengard-Ferrando should replace Hockney-Eastwood,” *arXiv:2103.08531 [physics]*, Mar. 2021.
- [8] M. Frey, A. Vinciguerra, S. Muralikrishnan, S. Mayani, V. Montanaro, and A. Adelman, “IPPL-framework/ippl: IPPL-3.1.0,” Sept. 2023. URL: <https://github.com/IPPL-framework/ippl/tree/IPPL-3.1.0>.
- [9] H. Carter Edwards, C. R. Trott, and D. Sunderland, “Kokkos: Enabling manycore performance portability through polymorphic memory access patterns,” *Journal of Parallel and Distributed Computing*, vol. 74, July 2014. Institution: Sandia National Lab. (SNL-NM), Albuquerque, NM (United States) Number: SAND-2013-5603J Publisher: Elsevier.
- [10] S. Muralikrishnan, M. Frey, A. Vinciguerra, M. Ligotino, A. J. Cerfon, M. Stoyanov, R. Gayatri, and A. Adelman, “Scaling and performance portability of the particle-in-cell scheme for plasma physics applications through mini-apps targeting exascale architectures,” in *Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing (PP)*, Proceedings, pp. 26–38, Society for Industrial and Applied Mathematics, Jan. 2024.
- [11] R. D. Ryne, “On FFT-based convolutions and correlations, with application to solving Poisson’s equation in an open rectangular pipe,” *arXiv:1111.4971 [physics]*, Nov. 2011. arXiv: 1111.4971.

- [12] R. D. Budiardja and C. Y. Cardall, “Parallel FFT-based Poisson solver for isolated three-dimensional systems,” *Computer Physics Communications*, vol. 182, pp. 2265–2275, Oct. 2011.
- [13] K. J. Raffenetti, “MPICH 4.1.2 released | MPICH.” URL: <https://www.mpich.org/2023/06/08/mpich-4-1-2-released/>, Accessed 09/01/2024.
- [14] X. Sáez, A. Soba, J. M. Cela, E. Sánchez, and F. Castejón, “Particle-in-Cell Algorithms for Plasma Simulations on Heterogeneous Architectures,” in *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 385–389, Feb. 2011. ISSN: 2377-5750.
- [15] J. P. Verboncoeur, “Particle simulation of plasmas: review and advances,” *Plasma Physics and Controlled Fusion*, vol. 47, pp. A231–A260, Apr. 2005. Publisher: IOP Publishing.