Scaling and performance portability of the particle-in-cell scheme for plasma physics applications through mini-apps targeting exascale architectures*

Sriramkrishnan Muralikrishnan† Matthias Frey‡ Alessandro Vinciguerra§
Michael Ligotino§ Antoine J. Cerfon¶ Miroslav Stoyanov‖ Rahulkumar Gayatri**
Andreas Adelmann††

Abstract

We perform a scaling and performance portability study of the electrostatic particle-in-cell scheme for plasma physics applications through a set of mini-apps we name "Alpine", which can make use of exascale computing capabilities. The mini-apps are based on IPPL, a framework that is designed around performance portable and dimensionality independent particles and fields. We benchmark the simulations with varying parameters, such as grid resolutions (512^3 to 2048^3) and number of simulation particles (10^9 to 10^{11}), with the following mini-apps: weak and strong Landau damping, bump-on-tail and two-stream instabilities, and the dynamics of an electron bunch in a charge-neutral Penning trap. We show strong and weak scaling and analyze the performance of different components on several pre-exascale architectures, such as Piz-Daint, Cori, Summit, and Perlmutter. While the scaling and portability study helps to identify the performance critical components of the particle-in-cell scheme on the current state-of-the-art computing architectures, the mini-apps by themselves can be used to develop new algorithms and optimize their high performance implementations targeting exascale architectures.

1 Introduction

Heterogeneous computing architectures are unavoidable as scientific computing moves toward the era of exascale computing. This is already evident from some of the supercomputers listed in Table 1, which consist of different types of CPUs and GPUs.

The programming paradigms or languages used with these architectures can differ significantly from one another. As a result, naively written codes could require rewrites of considerable portions of the codes simply to achieve compatibility with a given architecture, to say nothing of efficiency. In this context, performance portability is a key criterion for current and future simulation codes.

In the plasma physics community, particle-in-cell (PIC) schemes have been widely used for the simulation of kinetic plasmas since their inception [1, 2, 3]. The attractive features of PIC schemes include simplicity, ease of parallelization, and robustness for a wide variety of physical scenarios. Because of their flexibility and versatility, PIC schemes are employed in many production level plasma simulation and particle accelerator codes, such as TRISTAN-MP [5, 6], ORB5 [7], XGC [8], OSIRIS [9], IMPACT-T [10], OPAL [11] and Warp-X [12], to name a few.

Some of these production codes have already begun their journey toward performance portability as evidenced in [13, 14]. It is expected that a lot more will also do so in the near future in order to benefit from the high performance of current and future advanced computing architectures. Mini-applications (or mini-apps) which are performance portable can greatly help in this respect [15]. These are light-weight proxy codes which contain performance critical components of the application codes. They can be used for implementing new algorithms, optimizing implementations, and providing reliable performance expectations for the real application of interest on different computing architectures [15]. As such, mini-apps serve as an interface between applied mathematics, high performance computing, and production codes.

Our objective in this work is to study the performance portability and scaling of the components of electrostatic PIC schemes through a set of mini-apps

*The full version of the paper can be accessed at `https://arxiv.org/abs/2205.11052`

†Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany.
‡Mathematical Institute, University of St Andrews, KY16 9SS, UK.
§ETH Zurich, Switzerland.
¶Courant Institute of Mathematical Sciences, New York University, New York NY 10012, USA.
‖Oak Ridge National Laboratory, Oak Ridge, USA.
**National Energy Research Scientific Computing Center, Berkeley, CA, USA.
††Paul Scherrer Institut, Forschungsstrasse 111, 5232 Villigen, Switzerland.

Name	Location	CPUs	GPUs
Perlmutter	NERSC, USA	AMD Milan	NVIDIA A100
Summit	ORNL, USA	IBM POWER9	NVIDIA V100
Sunway TaihuLight	NSCC-Wuxi, China	Sunway SW26010	none
Fugaku	RIKEN, Japan	ARM A64FX	none
Frontier	ORNL, USA	AMD EPYC	AMD Instinct
Aurora	ANL, USA	Intel Xeon	Intel Xe

Table 1: An incomplete list of some of the extreme scale computing systems showing the diversity of the HPC landscape.

(Alpine[1]) with applications in plasma physics and particle accelerator modeling, targeting exascale architectures. In the present article, we consider physical situations for which the electrostatic assumption is justified and collisions can be neglected. In future work, we plan to extend the study by enriching the collection of mini-apps with electromagnetic examples, which may or may not include collisions and electron pair production. The mini-apps are built using the performance portable library Independent Parallel Particle Layer (IPPL), which is sketched in Section 2. The contributions of the current work are summarized below:

- Alpine provides a test bed for implementing new algorithms and/or novel implementations of existing algorithms related to PIC schemes in the context of exascale architectures in a performance portable way.
- The performance study in this work provides important insights on how different components of electrostatic PIC schemes function on different architectures.
- This work serves as a guidance for the plasma PIC community to identify the major causes of performance bottlenecks and better prepare for exascale architectures.
- So far, portable exascale PIC studies have mostly been conducted for PIC schemes designed for electromagnetic plasma models [14, 16]. To the best of our knowledge, this is the first study which considers the performance of a PIC scheme for an electrostatic plasma model in the portable exascale context. The fundamental difference comes from the fact that we need to solve a Poisson equation in electrostatic PIC schemes, which is global and has relatively less scalability than the purely local explicit electromagnetic PIC schemes.

[1]ALPINE: A set of performance portable pLasma physics Particle-in-cell mINi-apps for Exascale

This paper is organized as follows. Section 2 describes the IPPL library. The electrostatic PIC scheme implemented in the mini-apps is described in Section 3. Section 4 describes the mini-apps along with their physical parameters and verification studies. The strong and weak scaling results, as well as the performance analysis of different components on four different pre-exascale architectures, are presented in Section 5. Finally, we summarize our results in Section 6 and propose directions for future work.

2 IPPL

The Independent Parallel Particle Layer (IPPL) is a C++ library that was developed about 20 years ago and was inspired by and partially based on POOMA [17]. The general framework is designed to enable the rapid development of Lagrangian, Eulerian, and hybrid Eulerian-Lagrangian schemes. IPPL uses the Message Passing Interface (MPI) paradigm to distribute fields and particles across multiple processes. It also makes use of expression templates [18] to speed up the computation of mathematical operations on field and particle data.

Besides revising IPPL to the latest C++20 standard, we replaced the core data structures of IPPL with Kokkos [19, 20] data structures to enable performance portability across various hardware architectures.

The container holding the particles is essentially a *struct of arrays* where each particle attribute is in principle a Kokkos View enhanced with expression templates. Note that the underlying data type of a particle attribute is not restricted to scalar types; hence vector attributes (e.g. position and velocity) themselves are *arrays of struct*. The communication of particles among processes follows a pack-send-receive-unpack strategy where all particle attributes are serialized (or packed) into a single buffer and then deserialized (or unpacked) on the receiving end. These communication buffers are obtained from a memory pool with a user specified over-allocation parameter to avoid frequent memory allocations and deallocations, which are costly on GPUs.

To ensure an even workload across all MPI processes, the particles are redistributed regularly using the orthogonal recursive bisection algorithm [21]. For this purpose, the particle densities are interpolated onto the grid, which is then divided into subregions, each with approximately the same number of particles. The particles are then passed to the MPI process that occupies the subregion into which they fall.

As with the particle attributes, the fields are essentially Kokkos Views enhanced with expression templates. To share field data across process domain boundaries, IPPL uses halo (guard or ghost) cells. Halo data, like particle data, are exchanged via buffers. The number of halo layers can be set for each field independently.

An interface to heFFTe [22] enables us to compute fast Fourier transforms (FFT) in a portable manner. For more details on IPPL the readers are referred to Section 2 in [34].

3 Particle-in-cell method

3.1 Vlasov-Poisson system We consider the nonrelativistic Vlasov-Poisson system with a fixed magnetic field and introduce the PIC method in that setting. The electrons are immersed in a uniform, immobile, neutralizing background ion population and the electron dynamics is given by

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \frac{q_e}{m_e} \left(\mathbf{E} + \mathbf{v} \times \mathbf{B}_{ext} \right) \cdot \nabla_{\mathbf{v}} f = 0, \tag{3.1}$$

where $\mathbf{E} = \mathbf{E}_{sc} + \mathbf{E}_{ext}$, and the self-consistent field due to space charge is given by

$$\mathbf{E}_{sc} = -\nabla \phi, \quad -\Delta \phi = \rho / \varepsilon_0 = \left(\rho_e - \rho_i \right) / \varepsilon_0. \tag{3.2}$$

In equation (3.1), $f(\mathbf{x}, \mathbf{v}, t)$ is the electron phase-space distribution and q_e, m_e, and ε_0 are the electron charge, mass, and permittivity of free space, respectively. The total electron charge in the system is given by $Q_e = q_e \int \int f \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{v}$, the electron charge density by $\rho_e(\mathbf{x}) = q_e \int f \mathrm{d}\mathbf{v}$, and the constant ion density by $\rho_i = Q_e / \int \mathrm{d}\mathbf{x}$. Throughout this paper we use bold letters for vectors and non-bold letters for scalars. The numerical methods and algorithms we discuss in this article can be easily applied to situations involving nonuniform external magnetic fields with finite curvature. On the other hand, physical systems for which relativistic or electromagnetic effects play a significant role will generally require different approaches, and the corresponding codes will perform differently on large scale high performance computing architectures.

The particle-in-cell method discretizes the phase space distribution $f(\mathbf{x}, \mathbf{v}, t)$ in a Lagrangian way by means of macro-particles (hereafter referred to as "particles" for simplicity). At time $t = 0$, the distribution f is sampled, which leads to the creation of the computational particles. Subsequently, a typical computational cycle in PIC consists of the following steps:

1. Assign a shape function - e.g. cloud-in-cell [2] - to each particle p and deposit the electron charge onto an underlying mesh. This is known as "scatter" in PIC.
2. Use a grid-based Poisson solver to compute ϕ by solving $-\Delta \phi = \rho / \varepsilon_0$ and differentiate ϕ to get the electric field $\mathbf{E} = -\nabla \phi$ on the mesh with appropriate boundary conditions.
3. Interpolate $\mathbf{E}$ from the grid points to particle locations $\mathbf{x}_p$ using the same interpolation function as in the scatter operation. This is typically known as "gather" in PIC.
4. By means of a time integrator, advance the particle positions and velocities using

$$\frac{d\mathbf{v}_p}{dt} = \frac{q_e}{m_e} \left(\mathbf{E} + \mathbf{v} \times \mathbf{B}_{ext} \right) |_{\mathbf{x}=\mathbf{x}_p}, \tag{3.3}$$

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{v}_p. \tag{3.4}$$

3.2 Numerical implementation In our implementation we use the non-dimensional form of the Vlasov-Poisson system and we follow the same non-dimensionalization as in [23] (see Table I). Now, we succinctly describe the numerical algorithms that are used for the four PIC steps enumerated in the previous section in all our mini-apps.

First, we do a random sampling of the particles based on the distribution function f using inverse transform sampling [24]. It is important to create the particles locally in a load balanced manner[2] on each MPI rank to reduce the otherwise high memory and communication costs.

We use a cell-centered grid and cloud-in-cell shape function for the interpolation from particles to grid and vice versa. We use periodic boundary conditions in all directions and, since our grid spacings in the x, y and z directions are constants (although potentially different in each direction), we use an FFT-based spectral solver for the computation of the potential and electric field from the charge density. The FFTs in the field solver are computed using the heFFTe library. For the time integration, we use the synchronized form of the Boris scheme as given in equations 4(a) - 4(c) of [25], which gives both the velocity and position of the particles at integer time steps.

[2] In this paper, when we mention load balanced or imbalanced configurations, it is only with respect to particles unless we explicitly mention the fields.

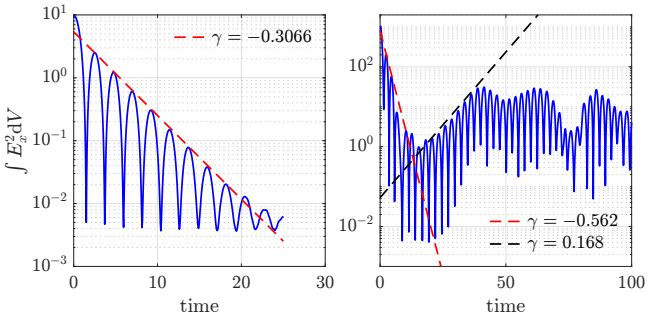


Figure 1: Landau damping of the electric field energy in the x-direction ($\int_V E_x^2 dV$, where V is the total simulation volume) as a function of time for weak (left) and strong (right) damping cases. The damping rates match well with the analytical values γ shown by the dashed lines as well as the results in [26]. The number of mesh points, number of particles and the time step in these simulations are 32^3; $83,886,080$ and 0.05 respectively.

4 The Mini-Apps

In this section, we briefly describe the electrostatic plasma physics problems that we consider for the scaling and performance study.

4.1 Landau damping The first problem we consider is Landau damping in the weak and strong damping regimes. It is a classical problem which has been studied extensively in the literature [4, 26, 27, 28, 29]. The availability of analytical results makes it an excellent candidate for our verification and performance study. Similar to [4], we consider the following initial distribution

$$f(t=0) = \frac{1}{(2\pi)^{3/2}} e^{-|\mathbf{v}|^2/2} \left(1 + \alpha \cos(kx)\right) \left(1 + \alpha \cos(ky)\right) \left(1 + \alpha \cos(kz)\right)$$

in the domain $[0, L]^3$, where $L = 2\pi/k$ is the length in each dimension. We choose the following parameters for our weak Landau damping tests: $k = 0.5, \alpha = 0.05$. The total electron charge based on our initial distribution is $Q_e = -L^3$. In the case of strong Landau damping, we use a stronger perturbation parameter $\alpha = 0.5$. The other parameters are the same as those for the weak damping case. The presence of a stronger perturbation parameter necessitates particle load balancing as will be shown in Figure 7. As a verification, we show in Figure 1 the damping of the electric field energy in the x-direction for the weak and strong damping cases. Our results agree well with the analytical rates as well as the results in [26].

4.2 Bump-on-tail/Two-stream instability The second mini-app we consider is the two-stream or bump-on-tail instability problem. Similar to Landau damping, this is another classical benchmark problem studied in the literature [26, 27, 28, 29], with analytical estimates for the growth rates, which can be calculated from the dispersion relation derived from the linearized equations.

We consider the following initial distribution of electrons

$$f(t=0) = \frac{1}{\sigma^3 (2\pi)^{3/2}} \left\{ (1-\epsilon)\, e^{-\frac{|\mathbf{v}-\mathbf{v}_{b1}|^2}{2\sigma^2}} + \epsilon e^{-\frac{|\mathbf{v}-\mathbf{v}_{b2}|^2}{2\sigma^2}} \right\} (1 + \alpha \cos(kz))$$

in the domain $[0, L]^3$, where $L = 2\pi/k$ is the length in each dimension. Depending on the choice of parameters, we get two flavors of this example. With $\epsilon = 0.5$, $\sigma = 0.1$, $k = 0.5$, $\alpha = 0.01$, $\mathbf{v}_{b1} = \{0, 0, -\pi/2\}$, and $\mathbf{v}_{b2} = \{0, 0, \pi/2\}$ we get the two-stream instability problem studied in [26]. On the other hand, choosing $\epsilon = 0.1$, $\sigma = 1/\sqrt{2}$, $k = 0.21$, $\alpha = 0.01$, $\mathbf{v}_{b1} = \{0, 0, 0\}$ and $\mathbf{v}_{b2} = \{0, 0, 4\}$ we get the bump-on-tail instability problem similar to [30]. The total charge Q_e is chosen in the same way as in the Landau damping example.

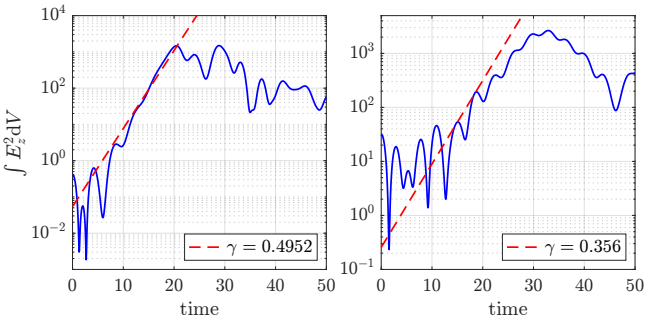


Figure 2: Electric field energy ($\int_V E_z^2 dV$, where V is the total simulation volume) in the z-direction as a function of time for the two-stream (left) and bump-on-tail instability (right) test cases. The growth rates agree well with the analytical values γ shown by the dashed lines as well as the results in [26]. The number of mesh points, number of particles and the time step in these simulations are 32^3; $83,886,080$ and 0.05 respectively.

In both cases, the electric field energy grows as a function of time as shown in Figure 2. Similar to Landau damping, we see very good agreement with the analytical rates as well as the results in [26]. Even though we simulate the bump-on-tail or two-stream instability in 3D-3V in our mini-app, the essential physics for the

initial distribution selected occurs predominantly in the z-direction, at least for early times.

4.3 Electron dynamics in a charge neutral Penning trap Our next mini-app corresponds to the dynamics of electrons in a Penning trap with a neutralizing static ion background, as in [31]. This problem involves bunching of electrons in the configuration space and, therefore, presents challenges in terms of field and particle load balancing. The initial conditions for this example, as well as the electron dynamics they lead to, are very similar to that of cyclotrons [32, 33, 31]. Since the particle accelerator library OPAL [11] will include the portable version of IPPL in the near future, this example is of interest from the point of view of cyclotron simulations.

Regarding the parameters for this problem, we follow the same setup as in [31]. The domain is $[0, L]^3$, where $L = 20$. The external magnetic field is given by $\mathbf{B}_{ext} = \{0, 0, 5\}$ and the quadrapole external electric field by

$$\mathbf{E}_{ext} = \left\{ -\frac{15}{L}\left(x - \frac{L}{2}\right), -\frac{15}{L}\left(y - \frac{L}{2}\right), \frac{30}{L}\left(z - \frac{L}{2}\right) \right\}.$$

For the initial conditions, we sample the phase space using a Gaussian distribution in all the variables. The mean and standard deviation for all the velocity components are 0 and 1, respectively. While the mean for all the configuration space variables is $L/2$, the standard deviations are $0.15L$, $0.05L$ and $0.2L$ for x, y, and z, respectively. The total electron charge is $Q_e = -1562.5$.

5 Scaling results

5.1 Setup In this section, we present representative strong and weak scaling results as well as the performance of different components for the mini-apps described in Section 4. We do not present the scaling results for the two-stream and bump-on-tail instabilities, as they are very similar to those of the weak Landau damping study; the timings differ by at most ±10 percent for the most part.

For the strong scaling, we consider the mesh and particle setups shown in Table 2. Both the cases have 8 particles per cell.

For the GPU simulations, we use 1 MPI rank per GPU and no OpenMP thread-based parallelism, whereas for the CPU simulations we use 1 MPI rank per node and take the number of OpenMP threads corresponding to the total number of CPU cores in that node. Basically, we use MPI for communication between GPUs both within a node and between nodes, whereas for CPU-based simulations, we use MPI only for communication between nodes while OpenMP-based shared memory parallelism is used within a node. This setup helps to minimize the particle and field communication costs. Further setup details regarding the reproducibility of the experiments along with the reason for the different choices is given in Appendix A.

Case	Grid (N_c)	Total number of Particles (N_p)
A	512^3	$1,073,741,824$
B	1024^3	$8,589,934,592$

Table 2: Cases considered for the strong scaling study with different number of grid points and particles.

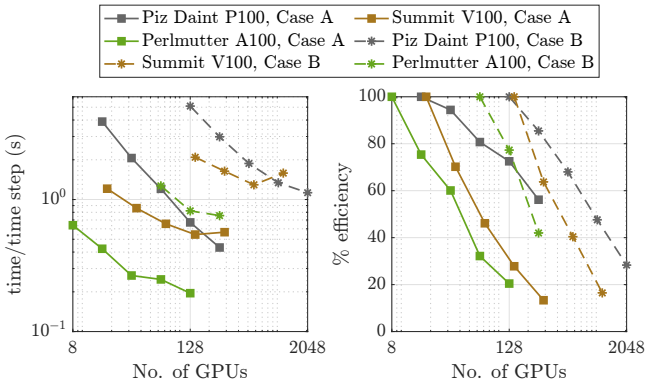
5.2 Performance comparison across different architectures In this section, we consider the weak Landau damping mini-app and evaluate its performance across different architectures. For this purpose we consider the CPU and GPU architectures in Table 3.

We utilize all the GPUs in each node with the GPU builds, whereas with the CPU builds we use 32 out of 36 and 64 out of 68 available cores per node on the Piz Daint and Cori systems, respectively. Multithreading is turned off for the CPU runs.

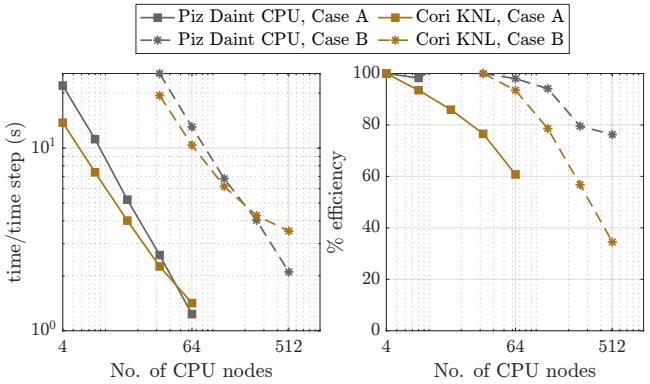
In Figure 3(a) the total time and efficiencies are compared across the three GPU architectures for the strong scaling study corresponding to cases A and B (cf. Table 2). In the case of Perlmutter, we can start the scaling study with half as many GPUs as the Piz Daint and Summit partitions, thanks to the higher memory configuration of A100 GPUs (40 GB) compared to those of P100 and V100 GPUs (16 GB). In terms of the absolute wall time per simulation time step, Perlmutter is the fastest, followed by Summit and then Piz Daint. In terms of scaling efficiencies, Piz Daint has the highest efficiency, followed by Perlmutter and then Summit. We can understand the total time and efficiencies better by looking at the cumulative computation and communication kernels (see Table 4 in Appendix A for the list of computation and communication kernels) across the three architectures for cases A and B in Figure 4(a). The computation kernels of Summit have a speedup of $3 - 4\times$ compared to Piz Daint until the scaling stops, whereas Perlmutter computation kernels have a speedup of roughly $10\times$ compared to Piz Daint. This is because of the architecture and CUDA compute capability of the GPUs: the latest A100 GPUs are more powerful than the V100s, which in turn are more powerful than the P100 GPUs. In terms of communication costs, Perlmutter has the lowest communication cost for both cases A and B. Comparing Summit and Piz Daint, for case A Summit has a higher communication cost than Piz

GPU architectures	CPU architectures
Piz Daint P100	Piz Daint (Intel Xeon E5-2695 v4 @ 2.1GHz)
Summit V100	Cori KNL (Intel Xeon Phi 7250 @ 1.4 GHz)
Perlmutter A100	

Table 3: List of CPU and GPU partitions used for the performance comparison across different architectures.



(a) Total time (left) and efficiency (right) on different GPU architectures



(b) Total time (left) and efficiency (right) on different CPU architectures

Figure 3: Comparison of different GPU architectures (top row) and CPU architectures (bottom row) with respect to strong scaling for the weak Landau damping mini-app.

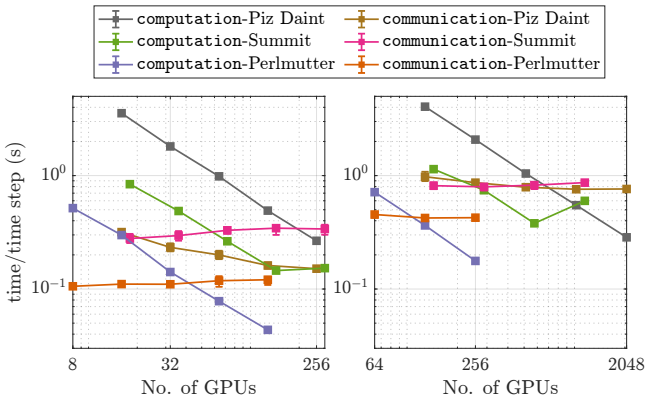
Daint, whereas for case B both are comparable. The higher communication cost of Summit is also reflected in the scaling of the computation kernels, where the field solver loses scaling earlier than on Piz Daint and Perlmutter.

We now consider the CPU architectures, and compare the strong scaling performance on Piz Daint and Cori KNL nodes. In Figure 3(b) we show the total time and efficiency for these two systems. For small node counts, the wall time per simulation time step on Cori nodes is slightly less than on Piz Daint nodes. However, the better scaling and efficiency of Piz Daint eventually leads to a lower runtime than Cori. From the computation and communication kernels in Figure 4(b) we notice that the computation kernels perform slightly better on Cori than on Piz Daint, but the communication time of Cori is higher, which eventually leads to a lower efficiency than Piz Daint. However, overall the performance per node of the two systems are very similar.

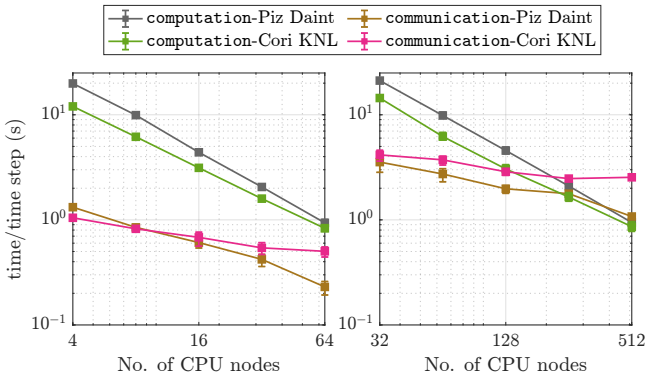
We refer the readers to [34] (Section 5.2.1) for the strong scaling and timings of individual computation and communication kernels on Piz Daint GPU and CPU architectures. The individual kernels on other GPU systems, i.e. Summit and Perlmutter, showed a similar scaling trend to Piz Daint. In terms of the timings, the field solve (`solve`) is the dominant computational kernel and the particles communication (`updateParticle`) is the dominant communication kernel. Their comparison with respect to the timings on Piz Daint is similar to that of the cumulative computation and communication kernels in Figure 4(a). The other computational kernels showed a maximum of up to 2× reduction with respect to the timings on Piz Daint whereas the field communication kernels are comparable to that of Piz Daint. On the CPU systems, the nodewise performance of individual kernels on Cori are very similar to that of the Piz Daint CPU nodes.

In Section 5.2.2 of [34] we considered two more cases with a higher number of particles per cell (64 as opposed to 8 considered here) and compared them with cases A and B.

5.3 Weak scaling We first describe the weak scaling setup for the weak Landau damping test problem before discussing the results. From now on, we will consider only the Piz Daint CPU and GPU architectures in Table 3. For GPU simulations, we take the base case for 1 GPU as a 256×128^2 grid, whereas for CPUs we take a 512×256^2 grid, because of the greater memory availability. For both CPUs and GPUs our simulations have 8 particles per cell. The maximum grid size and number of particles for the GPU simulations are $N_c = 2048^3$ and $N_p = 68,719,476,736$ on 2048 GPUs, whereas for CPU simulations they are $N_c = 4096 \times 2048^2$

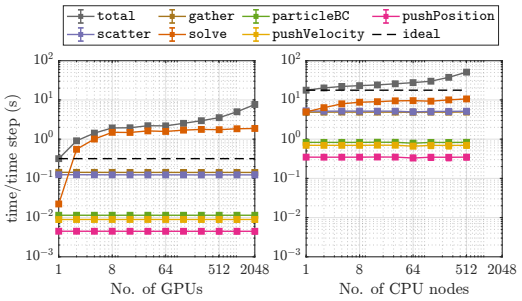


(a) Computation vs communication kernels for case A (left) and case B (right)

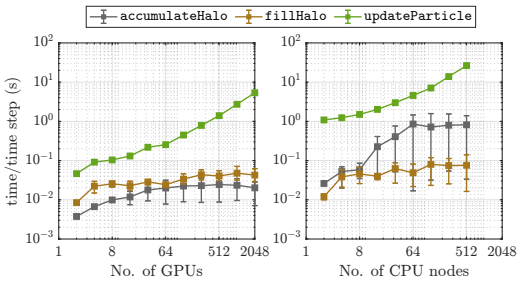


(b) Computation vs communication kernels for case A (left) and case B (right)

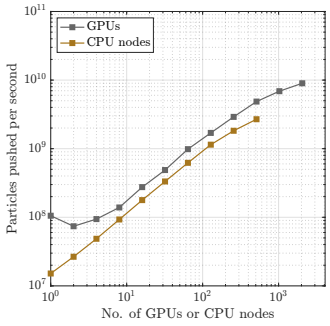
Figure 4: Timings of computation and communication kernels for different GPU architectures (top row) and CPU architectures (bottom row) with respect to strong scaling for the weak Landau damping mini-app.



(a) Computation kernels on GPUs (left) and CPUs (right)



(b) Communication kernels on GPUs (left) and CPUs (right)



(c) Particle push rate comparison between GPUs and CPUs

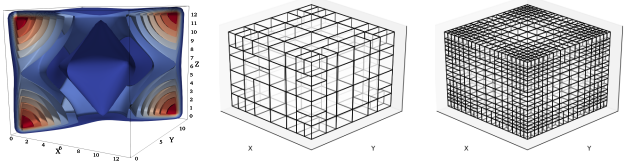
Figure 5: Weak Landau damping. Weak scaling of computation, communication kernels and particle push rate on GPUs and CPUs.

and $N_p = 137,438,953,472$ on 512 nodes (16,384 cores).

In Figure 5(a), we can see that the field solve is the dominant computation kernel. All the computation kernels scale ideally, except for the field solve and total runtime, as we have already seen for the strong scaling study due to the communication. The scaling of the field solver is also close to ideal starting from 8 GPUs or CPU nodes. This can be explained as follows. Starting from 8 MPI ranks, each process/core has a brick of field data and the number of transposes performed during the FFT remains the same. On the other hand, with 1, 2, or 4 ranks, we have either no communication or fewer transposes due to slab or pencil decompositions. Hence they take less time than on 8 ranks. The total time is mostly dictated by the field solve, except for the last few data points, where communication kernels dominate. We also see a linear increase asymptotically in the particle communication cost in Figure 5(b), due to the particle search and communication. In Figure 5(c), we show the number of particles pushed per second on GPUs and CPUs as measured in this weak scaling study. This metric is obtained by dividing the total number of particles by the wall time per simulation time step. We get a maximum of approximately 10^{10} particles per second on GPUs and 3×10^9 particles per second on CPUs.

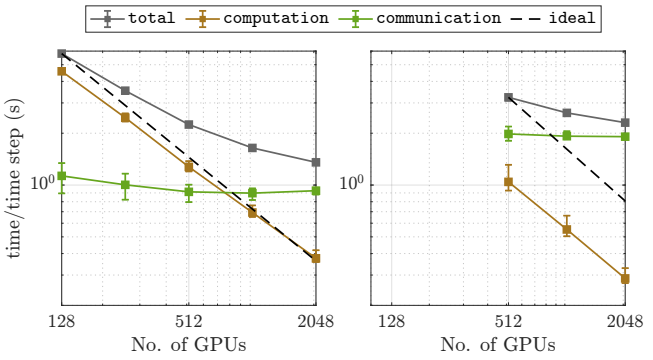
5.4 Comparison to electromagnetic PIC Electromagnetic (EM) PIC with explicit time integration

is purely local as the field equations are the hyperbolic Maxwell's equations. This is in contrast to the global Poisson solve involved in the electrostatic (ES) PIC which weakens its scalability and performance compared to EM PIC. Moreover, due to the CFL condition the time step size of the EM PIC should be such that the particles cannot travel more than one mesh cell. This helps to narrow the search of the particles during particle communication to only field neighbors. In ES PIC due to the lack of the CFL condition the time step size is only limited by the stability and accuracy of the time integrator used to push the particles and hence the particles can travel beyond the field neighbors. This calls for a global search which again limits the scalability. Because of these two reasons, the throughput as measured by particles pushed per second is usually orders of magnitude lower for ES PIC compared to EM PIC (cf. Table 1 in [35]).

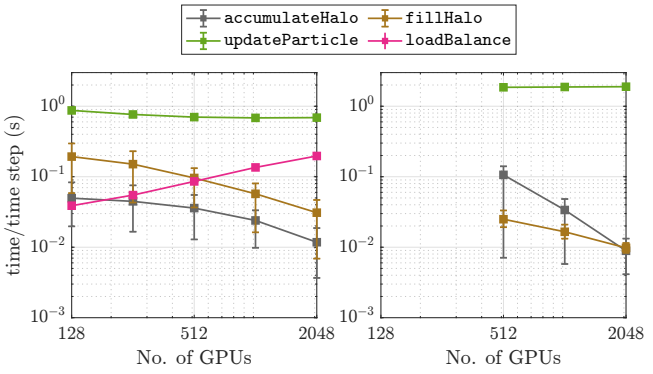


(a) Initial density (b) GPUs=128 (c) GPUs=2048

Figure 6: Strong Landau damping: Cross sectional view of the initial particle density profile showing high (red) and low (blue) densities, and particle load balanced domain decompositions for 128 and 2048 GPUs.



(a) Case B: Computation vs communication kernels with (left) and without (right) load balancing



(b) Case B: Communication kernels with (left) and without (right) load balancing

Figure 7: Strong Landau damping. Strong scaling on GPUs: comparison of scaling and performance with (left) and without (right) particle load balancing for case B.

5.5 Strong Landau damping We consider the case of strong Landau damping, which corresponds to a larger perturbation parameter α as compared to the weak damping case. This leads to a higher particle load imbalance among the MPI ranks with equal distribution of field domains. This in turn leads to an increased total time for the simulation compared to the load balanced case, as the MPI rank which takes the maximum time for the computation and communication kernels determines the overall simulation time. An even more important problem is the memory requirement, as the GPU or CPU node which has the greatest number of particles may not have enough memory to store them all, while the memory in the other GPUs or CPU nodes is under-utilized. Hence, particle load balancing is of critical importance in this example. We create the particles in a balanced way by means of orthogonal recursive bisection using the initial analytical density profile as shown in Figure 6(a). Figures 6(b) and 6(c) show the representative initial particle load balanced domain decompositions we obtain for 128 and 2048 GPUs. After the initial load balancing, we perform the balancing if the imbalance percentage given by

$$\text{imbalance}\% = \left(\frac{|n_{loc} - n_{ideal}|}{N_p} \right) \times 100$$

in any rank exceeds a given threshold. Here, n_{loc} is the local number of particles in each rank and $n_{ideal} = N_p/N_{ranks}$ is the ideal local number of particles. We set the threshold to 1% for this problem, as well as the Penning trap example in the next section, based on numerical experiments.

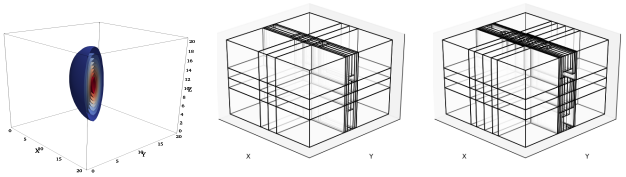
In Figure 7, we compare the results for case B with and without load balancing on GPUs. We do not show results for case A, as the scaling and performance of cases with and without load balancing are similar, and they are also comparable to the weak Landau damping results shown in Figure 3(a) and 4(a), due to the smaller total number of particles. We can see

from Figure 7(b) that the particle communication cost is higher in the case without load balancing. This leads to an earlier cross over point between computation and communication, and therefore loss of scaling as seen from Figure 7(a). This is because without load balancing some ranks contain a lot more particles than the others and also have to communicate more. The particle communication has a synchronization step, and since it has to wait for the last arriving rank, this leads to an increase in the `updateParticle` cost. Moreover, due to particle imbalance, the ranks which contain a lot of particles take more time in the computation kernels than the others, and this additional time also gets reflected in the synchronization steps, which in our case also correspond to `updateParticle`. We also note in the right column of Figure 7 that the case without load balancing is missing data points for 128 and 256 GPUs as these runs fail due to lack of memory. As explained before, this is due to the need for higher memory in some GPUs which have to store a lot of particles. We draw similar conclusions from the CPU simulations for case B and hence they are not shown.

Thus we can see from this example that our particle load balancing strategy is effective in cases with nonuniform particle distributions, without which the simulations either fail due to insufficient memory, or have poor scaling. We should, however, note that the load balancing strategy comes with a price. First, the additional overhead has to be small relative to the costs of the other kernels for it to be beneficial. This is the case in our tests shown in Figure 7 as, apart from the initial load balancing, the routine is never invoked for the set imbalance threshold of 1%. Second, our strategy of particle load balancing creates a load imbalance in the fields, as every rank now owns a brick of varied size in contrast to the uniform field distribution in the case without particle load balancing. This is visible from Figures 6(b) and 6(c). Although this does not affect the field solver scaling much in the case of strong Landau damping, for problems with highly nonuniform particle distributions, such as the Penning trap simulations, it can have a significant impact, as will be explained in Section 5.6.

The weak scaling results for the strong Landau damping simulations with load balancing are very similar to those of the weak Landau damping simulations shown in Figure 5. We therefore do not show them, to avoid repetition.

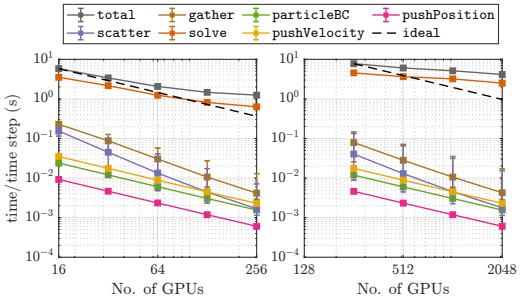
5.6 Electron dynamics in a charge neutral Penning trap In this section, we present the scaling results for the Penning trap mini-app. In Figure 8(a), the initial electron density is shown, which is a Gaussian



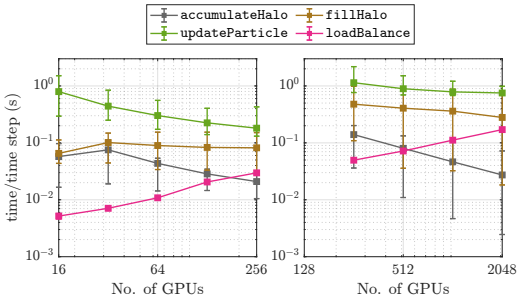
(a) Initial density (b) GPUs=256 (c) GPUs=2048

Figure 8: Penning trap: Cross sectional view of the initial particle density profile showing high (red) and low (blue) densities, and particle load balanced domain decompositions for 256 and 2048 GPUs.

bunch. In contrast to the Landau damping examples, this highly nonuniform particle distribution leads to a highly nonuniform field distribution after particle load balancing, as shown in Figures 8(b) and 8(c). In that respect the Penning trap mini-app is a harder test case in terms of field and particle load balancing compared to the Landau damping examples.



(a) Computation kernels for case A (left) and case B (right)



(b) Communication kernels for case A (left) and case B (right)

Figure 9: Penning trap. Strong scaling on GPUs: scaling of computation kernels, communication kernels, for cases A and B with load balancing.

In Figure 9, we show the strong scaling results for cases A and B with load balancing on GPUs. For this

example, we are unable to run most of the simulations without load balancing, due to memory requirements. We observe from Figure 9(a) that the field solver takes more time and that the scaling is much worse than for the Landau damping cases (see Section 5.2.1 and 5.2.2 in [34]). This is more significant in case B due to the large level of load imbalance in the fields, as shown in Figures 8(b) and 8(c). We also observe that even with particle load balancing, we are unable to run our simulation for case B with 128 GPUs due to insufficient memory for particle communication operations during time stepping. These observations clearly show that for highly nonuniform particle distributions, as in this Penning trap example, our current load balancing strategy has to be improved in order to effectively carry out these simulations with large numbers of particles on large numbers of nodes. We will investigate this in future work.

We notice from Figure 9(b) that even with particle load balancing, we have significant variation in timings across the GPUs for the `updateParticle`, which we did not observe in the strong Landau damping simulations. This again is a symptom of the highly nonuniform electron distribution in the Penning trap simulations.

For strong scaling study on CPUs as well as weak scaling studies on both GPUs and CPUs we refer the readers to Section 5.4 in [34]. Since there is little additional information to be inferred from these studies, they are omitted here.

5.7 Performance bottlenecks and guidance for the applications Based on the scaling and performance analysis from the mini-apps we have identified the following performance bottlenecks which could be useful for the applications.

1. In the electrostatic PIC applications with $\mathcal{O}(10)$ particles per cell the field solve is the most dominant computational kernel on both GPUs and CPUs and hence the overall time and the scalability of the application is mostly determined by the scalability and performance of the FFTs.

2. For highly nonuniform and clustered particle distributions particle load balancing is key, especially on GPUs, in order to carry out the simulations without facing memory issues. However, particle load balancing strategies such as orthogonal recursive bisection lead to a load imbalance in the fields and affects the scalability and performance. Thus novel load balancing strategies which take into account both memory and computational costs of the fields and the particles are needed in order to carry out these simulations efficiently.

6 Conclusion

In this work, we performed a scaling and performance portability study of the electrostatic particle-in-cell scheme for plasma physics applications by means of a set of mini-apps, namely "Alpine", targeting exascale architectures. The mini-apps include weak and strong Landau damping, the dynamics of an electron bunch in a quasi-neutral Penning trap, and the two-stream and bump-on-tail instabilities, which are commonly used as benchmarks for electrostatic PIC studies. Our scaling and performance analysis shows that the weak Landau damping simulations perform the best among the mini-apps in terms of scalability and time to solution. This is because the particle distribution is relatively uniform in this case, and the particle communication timings therefore remain small compared to the timings of the computation kernels. We obtained a maximum particle push rate of approximately 10^{10} particles per second on GPUs and 3×10^9 particles per second on CPUs in the weak scaling study. The scaling and performance of strong Landau damping simulations with particle load balancing are very similar to those of the weak Landau damping simulations, whereas without particle load balancing the particle communication costs as well as memory imbalance are much higher, which leads to poor scaling. The Penning trap mini-app corresponds to the toughest test case, due to the highly nonuniform particle distribution. In this case, particle load balancing leads to a significant imbalance in the fields, which then affects the scaling of the field solve. Our current load balancing strategy needs to be improved to handle such cases.

A performance comparison across different GPU and CPU architectures for the weak Landau damping mini-app shows that Perlmutter with the latest NVIDIA A100 GPUs performs the best in terms of wall time per simulation time step, with almost an order of magnitude speedup compared to the Piz Daint P100 GPUs, and approximately a three times speedup compared to Summit with V100 GPUs. In the comparison of CPU architectures, the wall time per simulation time step of Piz Daint and Cori KNL nodes are very similar, with Piz Daint showing better scaling than Cori.

In future work, we will optimize and improve upon the current load balancing strategy and particle communication. We will continue our benchmarking studies with Perlmutter and other upcoming architectures to test for higher numbers of particles, grid points, and GPUs and CPU cores. Finally, we would also like to extend our current study by adding more mini-apps to the Alpine collection, with test problems requiring the inclusion of collisions and the implementation of an electromagnetic PIC scheme.

Availability

Alpine and IPPL are open source projects. The sources can be downloaded from https://github.com/IPPL-framework/ippl.

Acknowledgments

The authors would like to thank the Kokkos team for helping us with all the queries during the development of IPPL and Alpine. We would like to thank Sonali Mayani for many fruitful discussions during the course of this project. We would also like to thank Marc Caubet Serrabou from PSI for his help with all the installations during the development of IPPL. This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 701647 and from the United States National Science Foundation under Grant No. PHY-1820852. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAPM888. We acknowledge access to Piz Daint at the Swiss National Supercomputing Centre, Switzerland under the PSI's share with the project IDs psi07 and psigpu. Finally, this research also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Appendix A. Further setup details for the scaling experiments

In terms of the time step, for the Landau damping mini-app we choose it to be proportional to the mesh size h as $\Delta t = 0.5h_{min}$. We also make sure that it is below the stable time step requirement of $\Delta t \leq 2\omega_{pe}^{-1}$, where ω_{pe} is the electron plasma frequency. Since the Penning trap simulations involve a lot more particle communication than the Landau damping problem, we choose the time step based on the finest mesh used in our scaling studies, i.e. $\Delta t = 0.5(L/2048) \approx 0.005$, just to have the same dynamics, and therefore similar particle communication, for different grid resolutions in a weak scaling study.

For the mini-apps we choose an over-allocation factor (in IPPL) of 2.0 for the weak and strong Landau damping tests, whereas for the Penning trap simulations we choose a value of 1.0 due to its high memory requirement per rank. These values are chosen based on numerical experiments.

For the ease of performance analysis, in Table 4 we split the significant kernels in our code into computation kernels, from which we can expect parallel efficiency, and communication kernels, which are required because of domain decomposition. However, this separation is not perfect since the FFTs required for the field solve are computed using heFFTe and this includes communication in addition to the purely local operations. This is because we use heFFTe as a black box for Fourier transforms and hence do not use IPPL timers inside its source code. The total time which is included in the computation kernels column is the time taken for the entire simulation to finish including the communication kernels. However, we exclude the time spent writing output data to files and an initial warm up call to the field solve. The first call to the field solve takes significantly more time than the subsequent ones owing to the initializations performed by heFFTe. In the communication kernels, the "Fill halo cells" and "Accumulate halo cells" operations are required during the gather and scatter stages of the PIC cycle, and the "Particle update" sends the particles to the appropriate ranks once they leave the local subdomain of the current rank. The components not included in Table 4 account for less than 10 percent of the total time in most of our simulations. We therefore do not consider them in the performance study.

We run the simulations for 20 time steps and report the wall time per simulation time step for each of the kernels in Table 4. As such, our performance figures are not indicative of production runs, as one typically needs to run for thousands of time steps in real plasma simulations. Furthermore, depending on the long time dynamics of the problem under consideration the performance can be significantly different. Our objective here is to assess the performance of different components in the mini-apps across different architectures without spending too many node hours or having to wait for long periods in the job submission queues.

The versions of the compilers, MPI, Kokkos, heFFTe, and IPPL used for the simulations on each of the computing architectures considered in this work are given in Appendix B. For the parameters in heFFTe we chose pencil decomposition, pipelined point-to-point communication with no-reordering. We refer the readers to [22] for descriptions of these parameters. These options are chosen based on our heFFTe benchmarking experiments with all the possible parameter combinations, selecting the best one in terms of scalability and time to solution. In terms of domain decomposition for our mini-apps, we use parallelization in all three directions for the fields, which gives each processor a brick of field data along with one layer of halo cells.

Computation kernels	Communication kernels
Gather (`gather`)	Particle update (`updateParticle`)
Scatter (`scatter`)	Fill halo cells (`fillHalo`)
Push position (`pushPosition`)	Accumulate halo cells (`accumulateHalo`)
Push velocity (`pushVelocity`)	Particle load balance (`loadBalance`)
Particle BCs (`particleBC`)	
FFT-based field solve (`solve`)	
Total time (`total`)	

Table 4: List of computation and communication kernels used for the performance study. Inside the parentheses are the labels which are used to represent the components in the figures.

Appendix B. Compilers and libraries used for the benchmarks on different computing architectures

We used Kokkos version 3.5.0 and heFFTe version 2.2.0 for all our simulations. The version of IPPL used for all the scaling studies is tagged `Scaling_study_for_Alpine_paper` in the repository. It can be obtained from https://github.com/IPPL-framework/ippl.

For each of the architectures, the compiler type and version and MPI version are specified in Table 5.

For the Piz Daint CPUs, we also conducted the benchmarking study with the following Intel compiler.

- `intel/2021.3.0`
- `cray-mpich/7.7.18`

The results are very comparable to those obtained with gcc. However, we sometimes observed an inconsistent "Bus error", the reason of which is still unknown and under investigation.

References

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

[2] C. K. Birdsall, A. B. Langdon, Plasma physics via computer simulation, CRC press, 2004.

[3] J. M. Dawson, Particle simulation of plasmas, Reviews of modern physics 55 (2) (1983) 403.

[4] L. F. Ricketson, A. J. Cerfon, Sparse grid techniques for particle-in-cell schemes, Plasma Physics and Controlled Fusion 59 (2) (2016) 024002.

[5] A. Spitkovsky, Simulations of relativistic collisionless shocks: shock structure and particle acceleration, in: AIP Conference Proceedings, Vol. 801, American Institute of Physics, 2005, pp. 345–350.

[6] O. Buneman, Computer space plasma physics, simulation techniques and softwares, ed, H. Matsumoto and Y. Omura (Terra Scientific, Tokyo, 1993) p 67.

[7] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T.-M. Tran, B. Mcmillan, O. Sauter, K. Appert, Y. Idomura, L. Villard, A global collisionless PIC code in magnetic coordinates, Computer Physics Communications 177 (5) (2007) 409–425.

[8] C.-S. Chang, S. Ku, Spontaneous rotation sources in a quiescent tokamak edge plasma, Physics of Plasmas 15 (6) (2008) 062510.

[9] R. A. Fonseca, L. O. Silva, F. S. Tsung, V. K. Decyk, W. Lu, C. Ren, W. B. Mori, S. Deng, S. Lee, T. Katsouleas, et al., OSIRIS: A three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators, in: International Conference on Computational Science, Springer, 2002, pp. 342–351.

[10] J. Qiang, S. Lidia, R. D. Ryne, C. Limborg-Deprey, Three-dimensional quasistatic model for high brightness beam dynamics simulation, Physical Review Special Topics-Accelerators and Beams 9 (4) (2006) 044204.

[11] A. Adelmann, P. Calvo, M. Frey, A. Gsell, U. Locans, C. Metzger-Kraus, N. Neveu, C. Rogers, S. Russell, S. Sheehy, J. Snuvernik, D. Winklehner, OPAL a versatile tool for charged particle accelerator simulations, arXiv preprint arXiv:1905.06654.

[12] J.-L. Vay, A. Almgren, J. Bell, L. Ge, D. Grote, M. Hogan, O. Kononenko, R. Lehe, A. Myers, C. Ng, et al., Warp-X: A new exascale computing platform for beam–plasma simulations, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 909 (2018) 476–479.

[13] S. M. Mniszewski, J. Belak, J.-L. Fattebert, C. F. Negre, S. R. Slattery, A. A. Adedoyin, R. F. Bird, C. Chang, G. Chen, S. Ethier, et al., Enabling particle applications for exascale computing platforms, The International Journal of High Performance Computing Applications 35 (6) (2021) 572–597.

[14] A. Myers, A. Almgren, L. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. P. Grote, M. Hogan, A. Huebl, et al., Porting WarpX to Gpu-accelerated platforms, Parallel Computing 108 (2021) 102833.

[15] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, R. W. Numrich,

Architecture	Compiler	MPI
Piz Daint GPU	`gcc/9.3.0`	`OpenMPI/4.1.2` with `CUDA/11.2`
Piz Daint CPU	`gcc/11.2.0`	`cray-mpich/7.7.18`
Cori KNL	`intel/19.1.1.217`	`cray-mpich/7.7.18`
Summit GPU	`gcc/9.1.0`	`spectrum-mpi/10.4.0` with `CUDA/11.0`
Perlmutter GPU	`gcc/11.2.0`	`OpenMPI/4.1.2` with `CUDA/11.4`

Table 5: Compiler type, version and MPI used for each of the architectures used for the scaling study in Section 5.

Improving performance via mini-applications, Sandia National Laboratories, Tech. Rep. SAND2009-5574 3.

[16] R. Bird, N. Tan, S. V. Luedtke, S. L. Harrell, M. Taufer, B. Albright, VPIC 2.0: Next generation particle-in-cell simulations, IEEE Transactions on Parallel and Distributed Systems 33 (4) (2021) 952–963.

[17] J. V. W. Reynders, J. Cummings, P. F. Dubois, The POOMA Framework, Computers in Physics 12 (5) (1998) 453–459.

[18] T. Veldhuizen, Expression templates, C++ Report 7 (5) (1995) 26–31.

[19] H. C. Edwards, C. R. Trott, D. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing 74 (12) (2014) 3202 – 3216, domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[20] C. R. Trott, D. Lebrun-Grandie, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, et al., Kokkos 3: Programming model extensions for the exascale era, IEEE Transactions on Parallel and Distributed Systems 33 (4) (2021) 805–817.

[21] D. J. Quinlan, M. Berndt, MLB: Multilevel load balancing for structured grid applications, Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (1997).

[22] A. Ayala, S. Tomov, A. Haidar, J. Dongarra, heFFTe: Highly efficient FFT for exascale, in: International Conference on Computational Science, Springer, 2020, pp. 262–275.

[23] D. Rodríguez-Patiño, S. Ramírez, J. Salcedo-Gallo, J. Hoyos, E. Restrepo-Parra, Implementation of the two-dimensional electrostatic particle-in-cell method, American Journal of Physics 88 (2) (2020) 159–167.

[24] L. Devroye, Nonuniform random variate generation, Handbooks in operations research and management science 13 (2006) 83–121.

[25] K. Tretiak, D. Ruprecht, An arbitrary order time-stepping algorithm for tracking particles in inhomogeneous magnetic fields, Journal of Computational Physics: X 4 (2019) 100036.

[26] A. Ho, I. A. M. Datta, U. Shumlak, Physics-based-adaptive plasma model for high-fidelity numerical simulations, Frontiers in Physics 6 (2018) 105.

[27] A. Myers, P. Colella, B. V. Straalen, A 4th-order particle-in-cell method with phase-space remapping for the Vlasov–Poisson equation, SIAM Journal on Scientific Computing 39 (3) (2017) B467–B485.

[28] K. Kormann, A semi-lagrangian vlasov solver in tensor train format, SIAM Journal on Scientific Computing 37 (4) (2015) B613–B632.

[29] G. Chen, L. Chacón, D. C. Barnes, An energy- and charge-conserving, implicit, electrostatic particle-in-cell algorithm, Journal of Computational Physics 230 (18) (2011) 7018–7036.

[30] S. Sarkar, S. Paul, R. Denra, Bump-on-tail instability in space plasmas, Physics of Plasmas 22 (10) (2015) 102109.

[31] S. Muralikrishnan, A. J. Cerfon, M. Frey, L. F. Ricketson, A. Adelmann, Sparse grid-based adaptive noise reduction strategy for particle-in-cell schemes, Journal of Computational Physics: X (2021) 100094.

[32] S. Adam, Space charge effects in cyclotrons-from simulations to insights, in: Proc. of the 14th Int. Conf. on Cyclotrons and their Applications,(World Scientific, Singapore, 1996), Vol. 446, 1995.

[33] J. Yang, A. Adelmann, M. Humbel, M. Seidel, T. Zhang, et al., Beam dynamics in high intensity cyclotrons including neighboring bunch effects: Model, implementation, and application, Physical Review Special Topics-Accelerators and Beams 13 (6) (2010) 064201.

[34] S. Muralikrishnan, M. Frey, A. Vinciguerra, M. Ligotino, A. Cerfon, M. Stoyanov, R. Gayatri, A. Adelmann, Scaling and performance portability of the particle-in-cell scheme for plasma physics applications through mini-apps targeting exascale architectures, arXiv:2205.11052, 2022.

[35] J. Xiao, J. Chen, J. Zheng, H. An, S. Huang, C. Yang, F. Li, Z. Zhang, Y. Huang, W. Han, et al., Symplectic structure-preserving particle-in-cell whole-volume simulation of tokamak plasmas to 111.3 trillion particles and 25.7 billion grids, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–13.