Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2020-0261

Vol. **x**, No. x, pp. 1-37
xxx 20xx

EMPIRE-PIC: A Performance Portable Unstructured Particle-in-Cell Code

Matthew T. Bettencourt[1], Dominic A. S. Brown[2,*], Keith L. Cartwright[1], Eric C. Cyr[1], Christian A. Glusa[1], Paul T. Lin[3], Stan G. Moore[1], Duncan A. O. McGregor[1], Roger P. Pawlowski[1], Edward G. Phillips[1], Nathan V. Roberts[1], Steven A. Wright[4], Satheesh Maheswaran[5], John P. Jones[6] and Stephen A. Jarvis[7]

[1] *Sandia National Laboratories, Albuquerque, NM.*
[2] *Department of Computer Science, University of Warwick, UK.*
[3] *Lawrence Berkeley National Laboratory, Berkeley, CA.*
[4] *Department of Computer Science, University of York, UK.*
[5] *Diamond Light Source Ltd, Diamond House, Harwell Science and Innovation Campus, Didcot, UK*
[6] *Atomic Weapons Establishment, Aldermaston, UK.*
[7] *College of Engineering and Physical Sciences, University of Birmingham, UK.*

Received 24 December 2020; Accepted (in revised version) 30 March 2021

Abstract. In this paper we introduce EMPIRE-PIC, a finite element method particle-in-cell (FEM-PIC) application developed at Sandia National Laboratories. The code has been developed in C++ using the Trilinos library and the Kokkos Performance Portability Framework to enable running on multiple modern compute architectures while only requiring maintenance of a single codebase. EMPIRE-PIC is capable of solving both electrostatic and electromagnetic problems in two- and three-dimensions to second-order accuracy in space and time. In this paper we validate the code against three benchmark problems – a simple electron orbit, an electrostatic Langmuir wave, and a transverse electromagnetic wave propagating through a plasma. We demonstrate the performance of EMPIRE-PIC on four different architectures: Intel Haswell CPUs, Intel's Xeon Phi Knights Landing, ARM Thunder-X2 CPUs, and NVIDIA Tesla V100 GPUs attached to IBM POWER9 processors. This analysis demonstrates scalability of the code up to more than two thousand GPUs, and greater than one hundred thousand CPUs.

AMS subject classifications: To be provided by authors

Key words: PIC, electrostatics, electromagnetics, HPC, performance portability.

*Corresponding author. *Email addresses:* `mbetten@sandia.gov` (M. Bettencourt), `Dominic.Brown@warwick.ac.uk` (D. A. S. Brown)

1 Introduction

High Performance Computing (HPC) provides huge benefit to the scientific community and has been especially useful within fields that require experiments that may be infeasible, and/or expensive to conduct physically. As a consequence of the continually rising computational performance of HPC systems, scientists are able to conduct research of ever increasing complexity. This improved capability will continue to grow with the move towards Exascale computing (the ability to carry out at least 10^{18} floating point operations per second), a major milestone in the field of HPC.

This significant improvement in performance has been accompanied by an increasing amount of diversity in modern compute architectures. For example, heterogeneous systems that make use of Graphics Processing Unit (GPU) accelerators or many-core CPUs are rapidly becoming more prevalent as we continue to move away from the traditional homogeneous cluster systems that were previously the norm [9]. As of June 2020, six out of the top ten supercomputers depend on heterogeneous architectures to achieve their compute performance [5]. The upcoming Exascale machines Aurora and Frontier will follow this same trend by using Intel and AMD GPUs respectively. As a result, it is now highly desirable for scientific codes to be able to perform well across a wide variety of systems, a concept often referred to as 'performance portability' [59]. However, this increase in architectural diversity brings greater difficulty in implementing production codes that are capable of fully exploiting the available hardware resources when compared to the traditional Single Program, Multiple Data (SPMD) MPI-based approach of the past. This problem is exacerbated by the fact that each architecture requires specific optimizations in order to achieve peak performance. Scientific codes must be able to adapt to this changing landscape without the difficulty of developing and maintaining several versions of the same application.

There are various standards that have been proposed to remedy this issue by providing directives to the compiler that sections of code should be run in parallel and/or on a given device – commonly an accelerator card. These include OpenMP [16] and OpenACC [31]. Another approach being considered to aid performance portability is the use of parallel programming frameworks or libraries. Examples include Kokkos [36], from Sandia National Laboratories (SNL), and RAJA [47], from Lawrence Livermore National Laboratory (LLNL), both of which make use of C++ template meta-programming to inject hardware-specific device code, targeting a system during compilation. Other notable examples of parallel programming frameworks include Khronos' OpenCL [2] and SYCL [42], and Intel's newly developed OneAPI [4].

HPC contributes to a variety of scientific fields, with the areas of fusion energy research, and the behavior of plasmas under various conditions being notable examples. Particle-in-Cell (PIC) [14, 32, 45, 55] codes are commonly used to carry out simulations of charged particles under the influence of electric and magnetic fields. Examples in fusion energy research include both Inertial Confinement Fusion (ICF) and Magnetic Confinement Fusion (MCF) devices. Such devices include the National Ignition Facil-

ity (NIF), located at LLNL, and the International Thermonuclear Experimental Reactor (ITER) located in France, which each attempt ICF and MCF respectively. Other applications include the behaviour of magnetrons in microwave generation systems, charged particle beams, laser-plasma interaction [7], astrophysical plasmas [63], and applications in biomedicine [38]. Traditionally, PIC employs a structured grid approach to represent the space being simulated, storing the field values on cell edges and faces [73], modelling particles as discrete objects in the problem space. The algorithm is commonly implemented using a Finite Difference Time Domain (FDTD) scheme with the electromagnetic fields represented on a staggered Yee grid, with discrete particles present within the domain [33, 51, 73]. Notable examples of such PIC codes include the Extendable PIC Open Collaboration (EPOCH) [7], OSIRIS [37], ICEPIC [15], the Plasma Simulation Code (PSC) [40] and VPIC [23, 24]. Gyrokinetic PIC algorithms have also been applied to the challenge of kinetic plasma simulation in five-dimensional phase space. Two such codes are GTC-P, the Gyrokinetic Toroidal Code [71], and XGC, the X-point Gyrokinetic Code, developed at Princeton University [50].

These codes are varied in their features and implementations but, with the exception of GTC-P and XGC, each operates on a traditional structured rectilinear grid. The application of such meshes to problems with high-fidelity geometries is challenging due to the extreme resolution that is needed. One approach to resolve this is through the use of Adaptive Mesh Refinement (AMR) to refine the problem only in areas of interest, reducing the total number of cells required for simpler sections of the geometry. The use of AMR-PIC has previously been explored for both electrostatic and electromagnetic problems by Vay et al. in WARP [69] and Warp-X [68], respectively.

An alternative solution to the problem of representing complex geometry is to use an unstructured computational mesh with finite elements of arbitrary shapes and sizes. Like AMR, this provides the flexibility of refining the problem in areas of key interest, but without the restriction that the grid cells themselves retain their structured properties. Examples of such PIC codes include PTetra [53] and the open-source Spacecraft Plasma Interaction Software (SPIS) [64].

As the PIC algorithm couples both grid-based and particle-based workloads, PIC codes exhibit a unique set of performance challenges to application developers on both current and future architectures. Such performance challenges have previously been well studied by other authors across a variety of system architectures. The performance of the GTC-P code at scale has been demonstrated on a number of notable HPC systems, including Sequoia, Piz Daint, Titan and Tianhe-2 [6, 67, 72], performing well in terms of both strong and weak scaling on a variety of compute architectures. The representative code Mini-EPOCH has been used to explore novel optimizations to the main EPOCH code by Bird et al. [12].

The use of GPUs to accelerate the PIC algorithm has also been documented by multiple authors, showing good speedup relative to CPU implementations [30, 34]. One example, PIConGPU [28], consists of a CUDA [3] implementation of the structured PIC algorithm, demonstrating scalability across multiple GPU compute nodes. Additionally,

the performance of the EPOCH code has also been evaluated for accelerator architectures, showing promising results across GPU-based systems [11]. The XGC code has demonstrated weak scaling on nearly the full pre-Exascale Summit machine (over 24000 GPUs) at Oak Ridge National Laboratory [65] by using the Cabana particle algorithm library [66], which is built on top of Kokkos.

In this paper we present the ElectroMagnetic Plasma In Realistic Environments PIC code (EMPIRE-PIC), an unstructured PIC application written in C++, developed at SNL. The code is capable of carrying out both electrostatic and electromagnetic simulations in two- and three-dimensional spaces using a variety of mesh topologies. These include quadrilateral, hexahedral, triangular, and tetrahedral elements. Note that the use of quadrilateral and hexahedral elements is limited to orthogonal quadrilaterals/hexahedra in order to avoid nonlinear iterations for particle tracking. EMPIRE-PIC makes extensive use of the Trilinos library [43], and uses Kokkos [36] as its parallel programming model together with MPI, allowing the application to be deployed on a wide variety of modern compute architectures including CPUs using OpenMP, and GPUs using NVIDIA's CUDA while only consisting of a single codebase. Additionally, EMPIRE-PIC has the following features:

- Support for thermal, beam, and space-charge-limited (SCL) particle injection;
- First-order absorbing boundary conditions;
- Integration with Tempus [1] for multiple time integration schemes;
- Large diagnostic suite including, but not limited to: volumetric output of charge, current, particle moments, and fields as well as probes obtaining the same quantiles as a function of time at a single location.

The contributions of this work are the following:

- We present EMPIRE-PIC, an unstructured PIC application that is capable of both electrostatic and electromagnetic simulations in two- and three-dimensions, and describe its core algorithm in detail;
- We outline the implementation of the key kernels of EMPIRE-PIC using Kokkos to achieve portability across modern architectures employing OpenMP for CPUs and NVIDIA's CUDA for GPUs; and highlight the performance challenges of these kernels;
- We analyse the convergence of the code for both electrostatics and electromagnetics using representative benchmark problems, demonstrating second-order convergence for both problem classes;
- We demonstrate the performance of the code at the single node level and at scale on a diverse set of architectures including Intel Haswell CPUs, Intel Xeon Phi, ARM Thunder-X2 CPUs, and NVIDIA Tesla V100 GPUs attached to IBM POWER9 CPUs.

The remainder of this paper is structured as follows: Section 2 presents our Particle-in-Cell algorithm, and the underlying formulation; Section 3 details the implementation of the various key application kernels, and their performance challenges; Section 4 consists of an analysis of the convergence of the schemes implemented in the code; Section 5 contains a performance analysis of EMPIRE-PIC on various modern compute architectures and, finally, Section 6 concludes the paper.

2 Finite element method particle-in-cell

PIC algorithms are commonly used for the simulation of plasma dynamics. This is accomplished by numerically solving the Klimontovich equation that governs the time evolution of a given plasma [58]. A collection of N_p discrete particles is used to represent the plasma, each particle has an associated position $\vec{x}$, velocity $\vec{v}$, charge q, and mass m. These are used to approximate a probability distribution, f, as a series of delta functions, δ, across all particles i

$$f = \sum_{i=1}^{N_p} f_i = \sum_{i=1}^{N_p} \delta(\vec{x} - \vec{x}_i)\delta(\vec{v} - \vec{v}_i). \tag{2.1}$$

The probability distribution for each particle is updated via Newton's Law and the Lorentz force equation, where $\vec{B}$ and $\vec{E}$ are the magnetic and electric fields, respectively. These can be expressed as the updates shown in Eqs. (2.2) and (2.3)

$$\frac{d\vec{x}_i}{dt} = \vec{v}_i, \tag{2.2}$$

$$\frac{d\vec{v}_i}{dt} = \frac{q_i}{m_i}\left(\vec{E}(\vec{x}_i) + \vec{v}_i \times \vec{B}(\vec{x}_i)\right). \tag{2.3}$$

These equations can then be assembled into the Klimontovich equation for collisionless particle dynamics [35, 49]:

$$\frac{\partial f_i}{\partial t} + \frac{\vec{v}_i}{m} \cdot \nabla f_i + \frac{q_i}{m_i}\left(\vec{E}(\vec{x}_i) + \vec{v}_i \times \vec{B}(\vec{x}_i)\right)\frac{\partial f_i}{\partial \vec{v}} = 0. \tag{2.4}$$

With given magnetic and electric fields, this system can be used to fully describe the particle evolution, and Maxwell's equations can be used to couple the charged particles to the electric and magnetic fields. They consist of the following: Gauss' Law, the magnetic divergence constraint, Faraday's Law, and Ampère's Law. For clarity, these equations are given below in the form of differential equations. Here ρ and $\vec{J}$ are the charge and current densities, and ϵ_0 and μ_0 are the permittivity and permeability of free space, respectively:

$$\nabla \cdot \vec{E} = \frac{\rho}{\epsilon_0}, \tag{2.5}$$

$$\nabla \cdot \vec{B} = 0, \tag{2.6}$$

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E}, \tag{2.7}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\mu_0 \epsilon_0} \nabla \times \vec{B} - \frac{1}{\epsilon_0} \vec{J}. \tag{2.8}$$

Finally, the particles can be coupled back to Maxwell's equations via the charge and current densities defined below.

$$\rho = \sum_{i=1}^{N_p} q_i f_i, \tag{2.9}$$

$$\vec{J} = \sum_{i=1}^{N_p} q_i \vec{v}_i f_i. \tag{2.10}$$

2.1 The particle-in-cell method

Introduced by Birdsall and Dawson, the PIC method is a well established procedure for modelling the behaviour of charged particles in the presence of electric and magnetic fields [13, 33]. Discrete particles are tracked in a Lagrangian frame, while the electric and magnetic fields are stored on stationary points on a fixed Eulerian mesh. Therefore, the algorithm can be thought of as two coupled solvers where one is responsible for updating the electric and magnetic fields, and another updates the particles via the method of characteristics and calculates their charge/current contributions back to the grid. These are referred to as the field solver and the particle mover (sometimes called the particle pusher), respectively. Combining these solvers results in the main time loop of the core PIC algorithm that is composed of four key steps, summarized in Fig. 1. In short this consists of: solving for the field values on the computational mesh, weighting these values to determine the fields at particle locations, updating the particle velocities and positions, and depositing the particle charge/current to grid points.

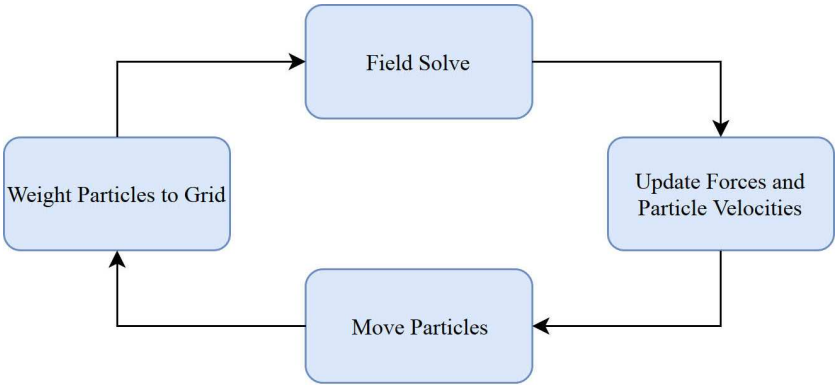


Figure 1: Flow chart summarising the key components of the PIC algorithm.

The number of physical particles (defined at the level of atoms and/or electrons) required to simulate a modern plasma system is exceedingly large. Thus, so-called super-particles are employed to make simulation via computation feasible. These can be thought of as 'computational particles' that reflect the behavior of a collection of physical particles. For example, one super-particle may represent many billions of electrons within a plasma. This allows the number of computational particles within the system to be much lower, therefore reducing the workload required. It should be noted that as the Lorentz force is only related to the charge-to-mass ratio of particles, these super-particles will exhibit the same collective motion as their physical counterparts. However, they will affect the fields by an amount proportional to the chosen weight.

2.1.1 Solving Maxwell's equations

In order to obtain the values of the electric and magnetic fields it is necessary to solve Maxwell's equations. Maxwell's equations hold true for all cases, but approximations can be made if certain conditions are satisfied. Specifically, if the movement of the plasma particles is slow in comparison to the speed of light, c, the equations can be reduced such that only Gauss' Law (Eq. (2.5)) must be solved, and the magnetic divergence constraint (Eq. (2.6)) is implicitly maintained. This is known as the electrostatic approximation, i.e., where the electric field is irrotational: $\nabla\times\vec{E}=0$. However if the current density is large or the particles move at relativistic velocities, then the electrostatic approximation is no longer valid. In such cases we must solve the full set of Maxwell's equations in order to account for the changing magnetic field. We refer to this class of problems as electromagnetic.

We now show in detail the formulation of both electrostatic and electromagnetic problems used in EMPIRE-PIC, such that they can be solved via the Finite Element Method (FEM) [48]. While the intention is not to be overly formal, the definition of several spaces is useful:

$$
\begin{aligned}
H_{Grad} &= \{v\in H^1(\Omega): v=0 \text{ on } \partial\Omega\},\\
H_{Curl} &= \{\vec{v}\in L^2(\Omega): \nabla\times\vec{v}\in L^2(\Omega),\ \vec{n}\times\vec{v}=0 \text{ on } \partial\Omega\},\\
H_{Div} &= \{\vec{v}\in L^2(\Omega): \nabla\cdot\vec{v}\in L^2(\Omega),\ \vec{n}\cdot\vec{v}=0 \text{ on } \partial\Omega\}.
\end{aligned}
$$

Note that here we have chosen spaces with simple Dirichlet condition assumptions. This is done for brevity of presentation, EMPIRE-PIC supports a range of electromagnetic and electrostatic boundary conditions as discussed above. One property of these spaces useful in proving implied involution conditions (see Eqs. (2.5) and (2.6)), is the natural nesting obtained from application of the appropriate derivative operator. With an abuse of notation, we have that:

$$
\nabla H_{Grad}\subset H_{Curl}, \quad \text{and} \quad \nabla\times H_{Curl}\subset H_{Div}. \tag{2.11}
$$

These properties are a trivial result of noting that $\nabla\times\nabla=0$ and $\nabla\cdot\nabla\times=0$. Once the problem has been formulated as described below, it can be solved using a variety of iterative or direct methods.

Electrostatic field solver

When conducting an electrostatic simulation we need only solve Eq. (2.5) during the field solve. In this case, the electric field can be represented as a gradient of electric potential, ϕ, as in Eq. (2.12), where $\vec{E}$ is defined as specified in Eq. (2.13)

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0}, \tag{2.12}$$

$$\vec{E} = -\nabla\phi. \tag{2.13}$$

Rewriting in the weak form, the electrostatic potential $\phi \in H_{Grad}$ satisfies

$$\int_\Omega \nabla\phi\cdot\nabla v \; d\Omega = \int_\Omega \frac{\rho}{\epsilon_0} v \; d\Omega \tag{2.14}$$

for all $v \in H_{Grad}$. Discretely, the domain Ω is decomposed into a finite element mesh that is used to define basis N_{node} functions $\hat{v}_i \in V_{Grad} \subset H_{Grad}$ where V_{Grad} is the space of piecewise linear functions (a so-called nodal discretization). Then the electrostatic potential is approximated by the linear combination

$$\phi \approx \sum_{j=1}^{N_{node}} \phi_j \hat{v}_j. \tag{2.15}$$

Substituting this approximation into the weak form, and testing against the discrete space V_{Grad} yields the linear system

$$S_{i,j} = \int_\Omega \nabla\hat{v}_j\cdot\nabla\hat{v}_i \; d\Omega, \tag{2.16}$$

$$\sum_{j=0}^{N} \phi_j S_{i,j} = \int_\Omega \frac{\rho}{\epsilon_0}\hat{v}_i \; d\Omega, \tag{2.17}$$

where it remains to determine the discrete charge density.

In order to handle the right hand side of the above expression, we approximate the charge density as

$$\rho(x) \approx \sum_{k=1}^{N_p} q_k \delta(\vec{x} - \vec{x}_k), \tag{2.18}$$

where $\vec{x}_k$ defines the location of N_p discrete particles. Thus the right hand side can be rewritten as

$$\int_0^L \frac{\rho}{\epsilon_0}\hat{v}_i \; d\Omega = \sum_{k=1}^{N_P} \int_0^L \frac{q_k\delta(\vec{x}_k)}{\epsilon_0}\hat{v}_i \; d\Omega = \sum_{k=1}^{N_P} \frac{q_k\hat{v}_i(\vec{x}_k)}{\epsilon_0}. \tag{2.19}$$

As a result of choosing a finite element basis, Eq. (2.17) results in a sparse matrix equation that can be solved through a variety of iterative or direct approximates when combined

with Eq. (2.19). With the approximation of ϕ determined, the electric field can be computed directly

$$\vec{E} \approx -\sum_{j=0}^{N} \phi_j \nabla \hat{v}_j. \tag{2.20}$$

Note that this approximation of $\vec{E}$ is irrotational by construction ($\nabla \times \nabla = 0$).

Electromagnetic field solver

For Maxwell's equations we also use a finite element discretization for space, while finite difference approaches are applied to discretize in time. A unique aspect of the transient Maxwell's equations is the addition of involution conditions satisfied for all time (see Eqs. (2.5) and (2.6)). There are a number of approaches to handle these conditions including divergence cleaning [52,54,60]. Our approach uses compatible discretizations [17, 18, 22, 44, 62] to ensure that Eq. (2.6) is satisfied strongly to machine precision, while satisfying Gauss' law using the discrete weak form in Eq. (2.14). We show below these conditions are satisfied by construction for all time under the assumption that the discrete form is satisfied at the initial time.

To this end, in 3D we use the lowest order Nédélec [57] elements $\vec{e} \in V_{Curl} \subset H_{Curl}$ (so-called edge elements), and Raviart-Thomas [25] elements $\vec{b} \in V_{Div} \subset H_{Div}$ (so-called face elements) to approximate the solution of the electromagnetic fields, thus

$$\vec{E} \approx \sum_{j=1}^{N_{edge}} E_j \vec{e}_j \quad \text{and} \quad \vec{B} \approx \sum_{j=1}^{N_{face}} B_j \vec{b}_j, \tag{2.21}$$

where N_{edge} is the number of edges, and N_{face} is the number of faces. Note that the basis functions evaluate to a 3D vector field, while the coefficients used in the expansion are scalars. These finite element spaces are constructed so that, along with the piecewise linear basis function approximating H_{Grad}, they satisfy the discrete equivalent of Eq. (2.11):

$$\nabla V_{Grad} \subset V_{Curl}, \quad \text{and} \quad \nabla \times V_{Curl} \subset V_{Div}. \tag{2.22}$$

In this way the basis functions are said to be compatible with the continuous spaces.

An immediate consequence of these choices is that our discretization of Faraday's law (Eq. (2.7)) is implemented in the strong form as

$$\sum_{j=1}^{N_{face}} \frac{\partial B_j}{\partial t} \vec{b}_j = -\sum_{j=1}^{N_{edge}} E_j \nabla \times \vec{e}_j \tag{2.23}$$

due to the relation $\nabla \times V_{Curl} \subset V_{Div}$. Further, taking the divergence of this expression gives

$$\sum_{j=1}^{N_{face}} \frac{\partial B_j}{\partial t} \nabla \cdot \vec{b}_j = -\sum_{j=1}^{N_{edge}} E_j \nabla \times \nabla \cdot \vec{e}_j = 0, \tag{2.24}$$

demonstrating the time evolution of the divergence of the discrete magnetic field is zero. Thus, if at the initial time the discrete magnetic field is divergence free, then the approximation of $\vec{B}$ is divergence free for all time. Thus, the selected discretization satisfies Eq. (2.6), the first of the two involutions, point-wise for all time as required (the time discrete version follows directly by replacing the continuous time derivative with the finite difference approximation and then applying an induction hypothesis on old time values).

Written in the weak form, Ampére's Law (Eq. (2.8)) defines the time evolution of $\vec{E} \in H_{Curl}$ such that

$$\int_\Omega \frac{\partial \vec{E}}{\partial t} \cdot \vec{w} \, d\Omega = \int_\Omega \frac{1}{\mu_o \epsilon_o} \vec{B} \cdot \nabla \times \vec{w} - \frac{1}{\epsilon_0} \vec{J} \cdot \vec{w} \, d\Omega \tag{2.25}$$

for all $\vec{w} \in H_{Curl}$ where $\vec{B} \in H_{Div}$ (choosing Dirichlet conditions for the spaces has simplified integration by parts for the sake of presentation. Despite this, EMPIRE-PIC does support a broad array of boundary conditions). Substituting the discrete spaces for their continuous analogs into Eq. (2.25) and including Eq. (2.23) leads to the semi-discrete form of the time evolution components of Maxwell's equations:

$$\frac{\partial B}{\partial t} = -CE, \tag{2.26}$$

$$M_E \frac{\partial E}{\partial t} = K_E B - J_{vec}, \tag{2.27}$$

where B and E represent discrete vectors of coefficients, and

$$M_E = \int_\Omega \hat{e}_i \cdot \hat{e}_j \, d\Omega, \tag{2.28}$$

$$K_E = -\frac{1}{\mu_0 \epsilon_0} \int_\Omega \hat{b}_j \cdot \nabla \times \hat{e}_i \, d\Omega, \tag{2.29}$$

$$J_{vec} = \int_\Omega \vec{J} \cdot \vec{e}_i \, d\Omega, \tag{2.30}$$

$$M_B = \int_\Omega \hat{b}_i \cdot \hat{b}_j \, d\Omega, \tag{2.31}$$

$$C = \nabla \times \vec{e}_i. \tag{2.32}$$

Here the matrix C has a non-zero entry if an edge is on the boundary of a given face in the mesh. That nonzero entry is ± 1 and orients a right-handed sum about the normal of the face – i.e. it encodes Stokes' theorem. In addition, we note the following matrix property arising from the compatible discretization

$$K_E = C^T M_B. \tag{2.33}$$

This will be useful in defining the linear solver below. How the current is specified by the particles is discussed in Section 2.4 below, as is the enforcement of the weak form of Gauss' law.

Considering the discretization of the time derivative for Faraday's and Ampère's Laws, EMPIRE-PIC includes backward Euler, Crank-Nicolson (C-N), and Friedman [39] time integration schemes. The C-N scheme is unconditionally A-stable, energy conservative, and second-order accurate. For n the discrete time level, we approximate the time derivative via a one-sided difference, and the f expression is evaluated at both the new and current time

$$\frac{\partial U}{\partial t} = f(U) \approx \frac{U^{n+1} - U^n}{\Delta t} = \frac{1}{2}\left[f(U^{n+1}) + f(U^n)\right]. \tag{2.34}$$

It should be noted that while C-N offers second-order convergence and exact energy conservation, it is not always an ideal integrator. The scheme is non-dissipative, so high frequency modes once perturbed will persist throughout the simulation. This, combined with the stochastic nature of particle simulations, can result in particle noise coupling to the fields, creating undesirable feedback [41]. In such a situation we generally recommend the Friedman integrator as it preferentially damps high frequency modes. For simplicity we present the C-N formulation for first-order Maxwell's equations:

$$B^{n+1} + \frac{\Delta t}{2} K_B E^{n+1} = B^n - \frac{\Delta t}{2} C E^n, \tag{2.35}$$

$$M_E E^{n+1} - \frac{c_0^2 \Delta t}{2} K_E B^{n+1} = M_E E^n + \frac{c_0^2 \Delta t}{2} K_E B^n - \frac{\Delta t}{\epsilon_0} J^{n+1/2}. \tag{2.36}$$

Here, we express the current as $J^{n+\frac{1}{2}}$, a vector form of $\frac{1}{\epsilon_0}\int_\Omega \vec{J}\cdot\hat{e}_i d\Omega$. If we assume that charge is represented by a point delta function with some charge q, we can define the current as shown in Eq. (2.37)

$$\begin{aligned} J_{vec} = \frac{1}{\epsilon_0}\int_\Omega \vec{J}\cdot\hat{e}_i \, d\Omega &= \frac{1}{\epsilon_0}\int_{n\Delta t}^{(n+1)\Delta t}\int_\Omega q\vec{u}(t)\delta(\vec{x}(t))\cdot\hat{e}_i d\Omega \, dt \\ &= \frac{1}{\epsilon_0}\int_{n\Delta t}^{(n+1)\Delta t} q\vec{u}(t)\cdot\hat{e}_i(\vec{x}(t)) \, dt. \end{aligned} \tag{2.37}$$

2.2 Weighting of fields to particles

After solving for the field values at the nodes of the simulation grid, we require a method of determining the value of the fields at any specific particle position. In order to do so we weight the field values from the grid nodes to the required location. Assuming that we know the values of the electric and magnetic fields we can evaluate the value of the fields at a given point as shown below by applying the basis function

$$\vec{E}(\vec{x}_i) = \sum_{j=0}^{N_{edge}} E_j \hat{e}_j(\vec{x}_i), \tag{2.38}$$

$$\vec{B}(\vec{x}_i) = \sum_{j=0}^{N_{face}} B_j \hat{b}_j(\vec{x}_i). \tag{2.39}$$

However, using the raw edge values of the electric field produces a large 'self-force' on the particle being pushed, as edge fields conserve energy whereas nodal fields conserve momentum [46]. The standard structured PIC algorithm uses special averaging of the edge fields to the nodes in order to generate the fields that are used to push the particles. In our unstructured algorithm, we mimic the approach used in the structured method by using projections from the edge-based electric fields to create fields that are based at the grid nodes. These projections have the form:

$$\int_\Omega \left(\vec{E}_{nodal}-\vec{E}_{edge}\right)\hat{v}\,d\Omega=\sum_{i=1}^{N_{node}}\vec{E}_{nodal,i}\int_\Omega \hat{v}_i\hat{v}\,d\Omega-\sum_{i=1}^{N_{edge}}E_{edge,i}\int_\Omega \vec{e}_i\hat{v}\,d\Omega=0 \quad \forall v\in V_{Grad}. \tag{2.40}$$

This is reduced to the solution of the linear system $M_{nodal}E_{nodal}=M_{nodal,edge}E_{edge}$ requiring inversion of the mass matrix for each dimension. One technique is to 'lump' the mass matrices, i.e., convert them into diagonal matrices with dual area on the diagonal. This leads to a volume-based field averaging that, when used on a uniform mesh, is identical to the method used by structured PIC simulations. However, the usefulness of this L^2 projection is mesh-dependent as the projection can introduce additional oscillations in the fields – especially on low-quality simplex meshes [61]. As such we provide the option of both direct interpolation of the edge and face fields and a nodal projection for the particle push.

2.3 Particle mover

The force felt by a charged particle due to the presence of electric and magnetic fields is described by the Lorentz force equation, shown in Eq. (2.41). The particle mover within EMPIRE-PIC is responsible for solving for this force on each particle within the simulation, and subsequently updating the particle velocities and positions. In our method we make use of the well-known Boris algorithm to handle the acceleration due to the electric field, and rotation about the magnetic field [21]

$$\vec{F}=q\left(\vec{E}+\vec{v}\times\vec{B}\right). \tag{2.41}$$

To make the algorithm relativistic we define $\vec{u}=\gamma\vec{v}$, where γ is the Lorentz factor. Then the velocity update satisfies

$$\frac{\vec{u}^{\,n+1/2}-\vec{u}^{\,n-1/2}}{\Delta t}=\frac{q}{m}\left[\vec{E}^{\,n}+\frac{1}{c}\frac{\vec{u}^{\,n-1/2}+\vec{u}^{\,n+1/2}}{2\gamma^n}\times\vec{B}^{\,n}\right]. \tag{2.42}$$

The Boris method separates the the electric field update from the magnetics rotation via the following substitutions:

$$\vec{u}^{\,n-1/2}=\vec{u}^{\,-}-\frac{q\vec{E}^{\,n}}{m}\frac{\Delta t}{2}, \tag{2.43}$$

$$\vec{u}^{\,n+1/2}=\vec{u}^{\,+}+\frac{q\vec{E}^{\,n}}{m}\frac{\Delta t}{2}, \tag{2.44}$$

$$\frac{\vec{u}^{\,+}-\vec{u}^{\,-}}{\Delta t}=\frac{q}{2\gamma^n mc}\left(\vec{u}^{\,+}+\vec{u}^{\,-}\right)\times\vec{B}^{\,n}. \tag{2.45}$$

Next we derive the expression for performing the rotation about the magnetic field. First we find the vector bisecting the angle formed between the velocity before and after the rotation. In a given time step, the velocity will rotate through the following angle: $\tan(\theta/2)=-(q\vec{B})\Delta t/2\gamma^n mc$. Rewriting this as a vector we obtain: $\vec{t}\equiv-\hat{b}\tan\theta/2$. We can then define $\vec{u}\,'$ as shown below:

$$\vec{u}\,'=\vec{u}^{\,-}+\vec{u}^{\,-}\times\vec{t}, \tag{2.46}$$

$$\vec{u}^{\,+}=\vec{u}^{\,-}+\vec{u}\,'\times\vec{s}, \tag{2.47}$$

$$\vec{s}=\frac{2\vec{t}}{1+\vec{t}^{\,2}}. \tag{2.48}$$

Once we have calculated $\vec{u}^{\,+}$, we can obtain the new particle velocity by adding an additional half of the acceleration as per Eqs. (2.43)-(2.45).

2.4 Weighting of particles to grid

During each time step in the PIC algorithm we must weight the contributions of each particle back onto the grid before we can commence the next field solve, though this contribution depends on whether the simulation is electrostatic or electromagnetic. For electrostatics, we apply Eq. (2.19) at the end of the particle move at the newly updated particle position. An electromagnetic simulation requires us to evaluate the current Eq. (2.37) as shown in Eq. (2.49). For simplices, it is sufficient to use a midpoint rule for the integration; however, for higher-order elements we must evaluate this temporal integration with higher-order numerical cubature. Specifically, in EMPIRE-PIC we use two-point Gaussian quadrature with points at $\left(1\pm1/\sqrt{3}\right)/2$, each with a weight of $1/2$ when using non-simplex elements. This reduces to the charge-conservation scheme for regular hexahedral elements presented by Villasenor and Buneman [70]

$$\int_{\Omega_j}\vec{J}\cdot\hat{e}_i\,dV=\sum_{k=1}^{N_P}\int_{\Omega j}\frac{1}{\Delta t}\int_{n\Delta t}^{(n+1)\Delta t}q_k\vec{u}_k\cdot\hat{e}_i\,dV=\sum_{k=1}^{N_P}\Delta t q_k\vec{u}_k\left(\vec{x}_k^{\,n+1/2}\right)\cdot\hat{e}_i\left(\vec{x}_k^{\,n+1/2}\right). \tag{2.49}$$

Demonstrating that the discrete weak form of Gauss' law is enforced is more involved than for the solenoidal condition shown above. Using the relation $\nabla V_{Grad}\subset V_{Curl}$, satisfaction of Eq. (2.25) for all $w\in V_{Curl}$ using the discrete time derivative implies that

$$\int_\Omega\frac{\vec{E}^{n+1}-\vec{E}^n}{\Delta t}\cdot\nabla v\,d\Omega=-\frac{1}{\epsilon_0}\int_\Omega\vec{J}\cdot\nabla v\,d\Omega\quad\forall v\in V_{Grad}. \tag{2.50}$$

The magnetic field term vanished because $\nabla\times\nabla v=0$. The right hand side describes the charge evolutions through space. Given a continuity expression that relates charge evolution to the divergence of current, an application of integration by parts will transform

Eq. (2.50) to a weak expression of Gauss' law Eq. (2.5). For more details, see for Miller et al. [56]. To show charge conservation with particle evolution the right hand-side of Eq. (2.50) is written using Eq. (2.49)

$$\int_\Omega \vec{J}\cdot\nabla v\, dV = \sum_{k=1}^{N_P} \frac{1}{\Delta t}\int_{n\Delta t}^{(n+1)\Delta t} q_k \vec{u}_k(t)\cdot\nabla v(x_k(t))\, dt. \tag{2.51}$$

To rewrite the right hand side, we recognize that the velocity evaluated at the midpoint is the time derivative of the position. Thus the chain rule can be applied to give a total time derivative:

$$\begin{aligned}\int_\Omega \vec{J}\cdot\nabla v\, dV &= \sum_{k=1}^{N_P}\int_{n\Delta t}^{(n+1)\Delta t} q_k \frac{dv(x_k(t))}{dt}\, dt \\ &= \frac{1}{\Delta t}\sum_{k=1}^{N_P} q_k\left(v({x_k}^{n+1}) - v({x_k}^{n})\right) \\ &= \frac{1}{\Delta t}\sum_{k=1}^{N_P}\int_\Omega q_k\delta(x-{x_k}^{n+1})v\, dV - \frac{1}{\Delta t}\sum_{k=1}^{N_P}\int_\Omega q_k\delta(x-{x_k}^{n})v\, dV.\end{aligned} \tag{2.52}$$

Assuming that weak Gauss' law is satisfied at $n=0$, then substituting the above expression into the right hand-side of Eq. (2.50) gives

$$\int_\Omega \vec{E}^n\cdot\nabla v\, dV = -\frac{1}{\epsilon_0}\sum_{k=1}^{N_P}\int_\Omega q_k\delta(\vec{x}-\vec{x}_k^{\,n})v\, dV \tag{2.53}$$

by induction on n.

2.5 Stability

The stability properties of EMPIRE-PIC follow those of the more traditional structured grid PIC algorithm. There are several Courant conditions that need to be met in order to achieve stability and accuracy. These constraints are summarized in Table 1, which provides values for stability and guidelines for accuracy. As we are using an implicit (Crank-Nicolson) electromagnetic field solver, EMPIRE-PIC is unconditionally stable. However, accuracy conditions continue to apply.

Table 1: Table of stability and accuracy criteria.

CFL	**Definition**	**Stability**	**Accuracy**
Speed of Light	$c\frac{\triangle t}{\triangle x}$	∞	$2\sim5$
Plasma Frequency	$\omega_p\triangle t$	2	~0.2
Cyclotron Frequency	$\omega_c\triangle t$	∞	~0.5
Advection	$v\frac{\triangle t}{\triangle x}$	∞	~1

3 Implementation

As systems continue to diversify it is crucial for modern HPC applications to be portable to multiple compute architectures, ideally from a single codebase. To this end an MPI+X approach has been taken for EMPIRE-PIC, using MPI for transferring data between distributed processes, and SNL's Kokkos performance portability library [36] for shared memory parallelism within a compute node. This enables EMPIRE-PIC to be used on any system for which Kokkos has a compatible backend. As a result of using the parallel constructs provided by Kokkos the application is portable across many modern architectures, on both single-node machines and at scale, using either OpenMP for CPU-based platforms or CUDA when executing on heterogeneous systems that employ NVIDIA GPU accelerators. Kokkos backends for the future Exascale machines, Aurora and Frontier, are also currently under development.

Currently within EMPIRE-PIC when running simulations across multiple MPI processes, the mesh is statically decomposed via a simple bisection method. This allows for an even distribution of mesh elements across all MPI processes. We use the Zoltan Trilinos package as a partitioner in order to perform this static decomposition [20], and the Teuchos Trilinos package to perform any MPI communications [43]. This can include the transfer of both mesh and particle data between neighbouring processes.

It is also important to consider the way that the data used by an application is laid out in memory. A memory layout that performs well on a CPU-based architecture is not guaranteed to perform well on GPUs. Kokkos abstracts the notion of memory layout away from the application developer by storing data in so-called 'views'. Each view has an associated memory layout template parameter, allowing the appropriate layout for a given architecture to be selected at compile time. CPU-based systems default to row-major layout such that a given thread can access consecutive data entries in order to make good use of cache. For GPUs a column-major layout is chosen as the default, such that consecutive threads in the same warp access consecutive locations in memory; this is known as coalesced access.

Views also have an assigned memory space that specifies where their data is stored. In the case of EMPIRE-PIC, host memory is used for CPU systems, whereas CUDA Unified Virtual Memory (UVM) is used for NVIDIA GPUs to allow for compatibility with CUDA builds of Trilinos and to reduce the need for explicit data transfers between host and device. Both the electric and magnetic field data are stored in $N \times 3$ Kokkos views, where N is the number of degrees-of-freedom (DOF) for the specific field. The particle data is stored in a structure-of-arrays (SoA) layout, using one-dimensional Kokkos 'dynamic views' that support constant-cost runtime resizing, greatly simplifying the addition of new particles to the data structure. The use of an SoA layout has the benefit of allowing each dynamic view to be accessed in unit stride, facilitating both vectorization on CPUs and coalesced access on GPUs. Efficient memory accesses are crucial for achieving high performance PIC in general, particularly for the particle-based kernels, due to the low arithmetic intensity relative to the amount of bytes moved to and from main memory.

Table 2: Table of approximate cost of the particle based kernels, in terms of FLOP/s and bytes read/written.

Kernel	FLOP Count	Bytes Read	Bytes Written
Accelerate	$106N_P$	$48N_P+48N_E$	$24N_P$
WeightFields	$128N_P$	$48N_P+308N_E$	$48N_P$
Move	$313N_P$	$84N_P+236N_E$	$172N_P$
Sort	$4N_S+29N_P$	$4N_S+200N_P$	$4N_S+110N_P$

This can be seen in Table 2, which shows approximate 'best case' counts of FLOP/s, bytes read, and bytes written for the particle kernels of an electromagnetic problem. Note that 'best case' assumes no particle migration occurs, particles are sorted at the beginning of the time-step, and that the particles are evenly distributed across an affine mesh. Here, N_P is the number of particles, N_E is the number of elements, and N_T is the number of particle types. N_S refers to the number of iterations carried out by a parallel scan, and is equal to $N_E N_T \log_2 N_{th}$, where N_{th} is the number of threads used for the scan.

3.1 Field solver

The formulation of both the electrostatic and electromagnetic problems is described in detail in Section 2.1.1. It is relatively simple to implement a linear field solver for the electrostatic formulation. To achieve this in EMPIRE-PIC we use the traditional Krylov iterative methods provided by the Belos Trilinos package [8]. The MueLu Trilinos package is used in conjunction with Belos to provide an Algebraic Multigrid (AMG) preconditioner [10]. The Belos and MueLu packages both make full use of MPI and Kokkos in order to achieve performance both on a single node and at scale. Additionally, the Trilinos packages Tpetra and Kokkos Kernels are used to provide linear algebra objects and portable interfaces to vendor-optimised implementations of common linear algebra operations, respectively.

The electromagnetic formulation requires a more specialized solver, detailed here. The C-N evolution expression for the electromagnetic fields shown in Eqs. (2.36) and (2.35) can be rewritten as a solution the linear system

$$\underbrace{\begin{bmatrix} I_B & \frac{\Delta t}{2}C \\ -\frac{c_0^2\Delta t}{2}K_E & M_E \end{bmatrix}}_{A} \begin{bmatrix} B^{n+1} \\ E^{n+1} \end{bmatrix} = \begin{bmatrix} B^n - \frac{\Delta t}{2}CE^n \\ M_E E^n + \frac{c_0^2\Delta t}{2}K_E B^n - \frac{\Delta t}{\epsilon_0}J^{n+1/2} \end{bmatrix}. \tag{3.1}$$

The above system can then be expressed in terms of the block LU decomposition

$$A = \begin{bmatrix} I_B & 0 \\ -\frac{c_0^2\Delta t}{2}K_E & I_E \end{bmatrix} \begin{bmatrix} I_B & \frac{\Delta t}{2}C \\ 0 & S_E \end{bmatrix}, \tag{3.2}$$

where the electric field Schur complement, S_E, is given by:

$$S_E = M_E + \frac{c_0^2 \Delta t^2}{4} K_E C. \tag{3.3}$$

The Schur complement in this case is sparse, moreover the operator $K_E C = C^T M_B C$ corresponds to a curl Laplacian. While one could assemble this term directly we compute it with matrix-matrix products as fewer quadrature evaluations are required to assemble the face basis mass matrix compared to the edge basis curl-curl matrix. Using the Schur complement we can reformulate the coupled, non-symmetric problem as a lower-triangular solve of the form

$$S_E E^{n+1} = \left(M_E - \frac{c_0^2 \Delta t^2}{4} K_E C \right) E^n + c_0^2 \Delta t K_E B^n - \frac{\Delta t}{\epsilon_0} J^{n+1/2}, \tag{3.4}$$

$$B^{n+1} = B^n - \frac{\Delta t}{2} C \left(E^{n+1} + E^n \right). \tag{3.5}$$

This reformulation of the field solve produces a discrete evolution equation with many algorithmic similarities to implicit vector wave formulations of Maxwell's equations while still offering the advantages of a single-step, first-order system formulation. For example, the Schur complement matrix S_E is symmetric positive-definite. As such the system can be solved with a preconditioned conjugate gradient method rather than applying preconditioned GMRES to the non-symmetric block system. This allows for reduced computational cost, for instance reducing the memory overhead of the field solve. In addition to simple preconditioners such as Jacobi, EMPIRE-PIC offers the AMG preconditioner for Maxwell's equations proposed by Bochev et al. [19]. A Kokkos implementation of this preconditioner is included in the Trilinos library MueLu, where it is referred to as "RefMaxwell."

3.2 Weighting fields to particles

As the values of the fields are only known at the points of the problem mesh, it is necessary to interpolate their values to the position of the particles as shown in Section 2.2. To accomplish this, the particle container contains both $\vec{E}$ and $\vec{B}$ arrays to store the results of gathering the fields to each particle. This also improves spatial locality during the particle move by avoiding the repeated reading of irregular memory locations that would result from computing the field values for each particle on the fly. As an additional optimization, we also employ loop fusion by merging the kernels that perform the magnetic and electric field weighting. This halves the number of times each particle must be fetched from main memory, and also eliminates redundant evaluation of the basis functions at the particle location. Furthermore, the operations of this kernel are free from dependencies, making the code much easier to parallelize. The code also contains a field-weighting kernel that instead makes use of the raw edge and face field values, should this be desired.

3.3 Particle mover

The particle mover updates the velocity and position of each particle based on the field values gathered during the field weighting step. This is done via the method detailed in Section 2.3. However, the velocity calculation differs for electrostatic and electromagnetic simulations. For electrostatics it is sufficient to determine the acceleration due to the electric field and subsequently update the particle velocity, as the rotation due to the magnetic field can be ignored. It is then trivial to update the particle position using the newly calculated velocity. In the case where a particle would cross an element boundary the move is broken up into its segments, and the move routine is applied to each segment in turn. In electromagnetic simulations each particle also deposits current to each element crossed during this step.

As the position update of each particle is completely independent from the movement of the others, this kernel also lends itself well to parallelization due to the lack of dependencies. This is especially apparent on GPU-based systems where an extremely large number of threads can be executed in parallel, combined with high-bandwidth memory. However, the additional control flow to handle particles crossing process and/or element boundaries can lead to warp divergence on GPUs and make achieving satisfactory vectorization challenging on CPU systems.

When crossing element boundaries it is also possible for particles to leave the domain handled by their current processor, therefore requiring migration to the destination processor. Once all local particles have been moved to completion, all particles that have been marked for communication are packed into send buffers in parallel before migration takes place. In order to reduce the overhead of waiting for MPI communications to take place, all migration operations are carried out using asynchronous (non-blocking) communications between processes. This allows both computation and communications to be interleaved. One such computation is compacting the particle arrays on each processor in order to remove the 'holes' where sent particles previously resided. This ensures that the particle arrays do not become fragmented in memory as the simulation progresses. Once communication has finished, newly arrived particles are unpacked and appended to the array held by the destination processor. Lastly, a collective reduction is used to determine the total number of particles migrated across all processes. The particle mover is then executed again to move the newly arrived particles. Subsequently, this process is repeated until all particles are finished moving for the current time-step, specifically when it is detected that no particles were migrated during an iteration of the move.

3.4 Particle sort

Once the particle move has completed for a given step the array of particles is then sorted firstly by cell and sub-sorted by species type. While this is useful for a variety of simulation diagnostics there are also performance-related benefits. Specifically, keeping particles that are near to each other in the simulation close in memory provides benefit in

Algorithm 1 Pseudocode for the particle sort.

```
procedure SORT(do_subsort)
    par_for_each particle do
        Increment destination bin count with atomic add
    end par_for_each
    par_scan i ← 0 to N_types · N_elem do
        Find the start index for each type with prefix sum
    end par_scan
    par_for_each particle do
        ▷ Find the new index for each particle
        Atomic fetch add bin start index
    end par_for_each
    par_for_each particle do
        Reverse mapping to get locations after sort
    end par_for_each
    if do_subsort = true then
        ▷ Sort each bin in parallel
        par_for_each i ← 0 to N_types · N_elem do
            UNROLLEDQUICKSORT(bin[i])
        end par_for_each
    end if
end procedure
```

terms of spatial memory locality. This allows data that would be shared across multiple particles, such as common grid data when weighting the fields to the particles, to remain in registers or cache for longer, thereby reducing the total number of memory loads and stores. This is especially beneficial due to the memory-bound nature of the particle kernels in a PIC code.

As the particles in EMPIRE-PIC are stored in a Structure of Arrays (SoA) layout, all of the arrays must be sorted to maintain correctness. Pseudocode for the particle sorter is shown in Algorithm 1. A key point is that initially only the particle index is sorted to produce a permutation vector, thus reducing the total amount of particle data that must be read from memory. Informally, the sorting procedure can be summarized as follows. First, a parallel for loop counts the number of each particle type within each element. A parallel scan performing a prefix sum then determines the starting offset of each bin in memory. This data is then used to determine the index of each particle in the new array, which is then used to create a reverse mapping. Then, the magnitude of each particle's position vector is stored in temporary working memory, and each bin is then sub-sorted by this value to generate a permutation vector. This sub-sort is done in parallel with one thread being assigned to handle each bin, where the bins are sorted via quicksort with unrolled recursion to reduce the stack space required on GPUs. The final generated

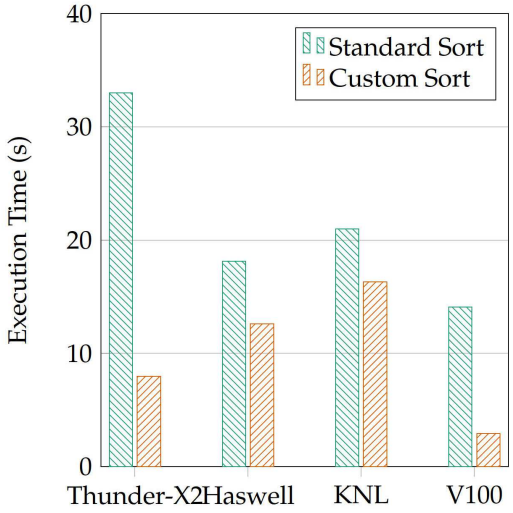


Figure 2: Performance comparison between particle sorters.

permutation vector is then applied to all relevant arrays within the particle container.

The performance of the sorter was compared to a standard stable sort on the systems shown in Table 4 for a single node. In the case of the GPU tests, a single V100 card was used. A 16 million particle, electromagnetic problem (size small (S) in Table 5) was used for the tests, with 100 time-steps being carried out. On the CPU systems, an OpenMP task-based parallel stable sort was used as the base case. For the V100 GPU, the stable sort provided by the Thrust library was chosen. The results of these tests are shown in Fig. 2, where it is clear to see that the specialized sort performs better across all four systems.

3.5 Weighting particles to grid

During each step of the main EMPIRE-PIC time loop, each particle must make contributions back to the grid prior to the next field solve. For electrostatic problems each particle must deposit charge to each node of its current cell at the end of the time step as shown in Eq. (2.19), while in electromagnetic simulations the particle must contribute current to each element crossed during the particle move, as defined in Eq. (2.49). By default, current is deposited to element edges and charge to element nodes, but nodal current can be specified if this is desired.

As there can be many particles occupying the same grid cell at any one time, there is the possibility of data hazards when executing these procedures in parallel. This occurs in the form of a write conflict when multiple threads attempt to deposit charge or current to the same memory location(s) simultaneously. Therefore, some method of protection is required in order to prevent erroneous results. Possible solutions to the data hazard problem include the use of colouring methods to ensure that threads write only to non-

Table 3: Particle move execution time for different write conflict resolution methods.

	Move Execution Time (s)		
Architecture	**No Atomics**	**Atomics**	**Scatter View**
Haswell	16.53	22.23	13.37
KNL	35.76	57.25	35.41
Thunder-X2	12.80	24.58	14.72
V100	3.65	3.91	3.91

conflicting locations, or keeping thread local copies of the data, only requiring an atomic operation or reduction for the final deposit.

We evaluated the performance of particle-to-grid weighting in EMPIRE-PIC for two different approaches, specifically atomic writes, and data replication with a follow-up reduction. Atomic writes were implemented through Kokkos' built-in atomic view access trait, while data replication was achieved via the Kokkos `ScatterView` construct. As the overhead of atomics on modern GPUs is low due to hardware acceleration, `ScatterView` continues to use atomic writes for CUDA builds, but switches to data replication for OpenMP builds. As with the sorter, the performance of the weighting was tested for a 16 million particle, electromagnetic problem on a single node of all systems. Again, in the case of the GPU tests, a single V100 card was used. Table 3 shows the time taken to complete the particle move on all systems for each approach. It is clear to see that the data replication approach used by `ScatterView` for OpenMP vastly outperforms atomic writes, while the performance on the V100 GPU remains the same as atomics are still used. Of particular interest is that the `ScatterView` implementation outperforms the version of the code where no conflict resolution is implemented for the Haswell and KNL systems. This is due to improved cache behaviour as a result of reduced false sharing penalties, as each thread now only operates on thread-local data instead of a shared global Kokkos view.

4 Numerical results

The following section examines both charge conservation and the convergence behaviour of the PIC algorithm implemented within EMPIRE-PIC. To this end we consider both electrostatic and electromagnetic simulations chosen to be representative of the problems that can be solved using the application. Specifically, an electrostatic electron orbit, electrostatic Langmuir Wave, and a Transverse Electromagnetic (TEM) wave propagating through a plasma are used for convergence studies.

4.1 Charge conservation

In Section 2.4 we documented the expectation of charge conservation for EMPIRE-PIC. A simple test was developed to demonstrate this property. Specifically, two metal surfaces

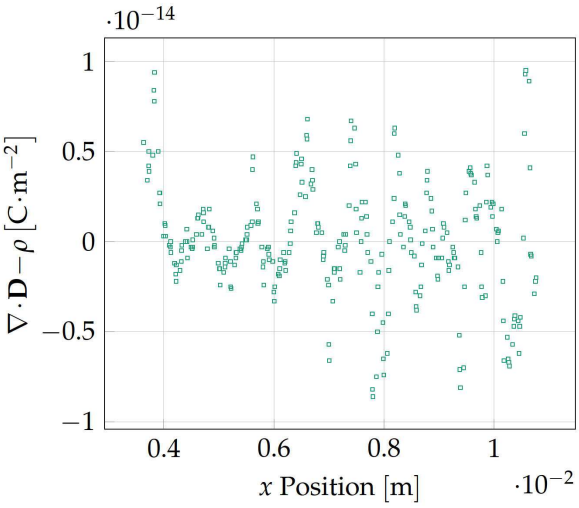


Figure 3: Data showing that EMPIRE-PIC conserves charge to within round-off error throughout the simulation space.

were separated by 11mm and were initialized with a 1MV/m electric field. The lower surface was allowed to emit with a space charge limited (SCL) emission algorithm which was designed to drive the electric field towards zero, thus causing a pulse of charge to traverse the gap. We plot $\nabla \cdot \mathbf{D} - \rho$ at 0.2 ns in space in Fig. 3, which shows the lack of charge conservation to be within the round-off level – as we would expect from EMPIRE-PIC.

4.2 Electrostatic electron orbit

We now consider the behaviour of our algorithm on a very basic electrostatic problem, consisting of a stationary H^+ ion being orbited by a single electron for one period. The particles are situated on a distorted mesh of tetrahedral elements, the ion positioned at the centre, and the electron has an orbit radius of $r_{orbit} = 5.291 \times 10^{-8}$ m. The length of the domain in both x and y directions is equal to $1.5 \times r_{orbit}$. We also specify the problem boundary conditions to an analytical value defined as the exact value of the potential at the boundary: $\phi = \frac{q}{2\pi\epsilon_0} \ln\left(r^{-1}\right)$.

The initial conditions of the problem can be derived as follows. Given the electrostatic assumption, we can reduce the Lorentz force to $\vec{F} = q\vec{E}$. Then, from centripetal force and Gauss' Law we can write the following to obtain an expression for the electric field:

$$qE(r) = m\omega^2 r, \tag{4.1}$$

$$\int \vec{E} \cdot \hat{n}\, dA = \int_V \frac{\rho}{\epsilon}. \tag{4.2}$$

We are using an electron that does not deposit charge to the mesh in order to simplify the boundary conditions, due to having zero charge, but finite charge-to-mass ratio. There-

fore, as the fields are not changed, the above can be simplified. We can now rewrite and substitute Eqs. (4.1) and (4.2) in order to derive an expression for the angular velocity ω, and also velocity v which can then be resolved into its x and y components

$$E(r) = -\frac{1}{2\pi r}\frac{q}{\epsilon_0}, \tag{4.3}$$

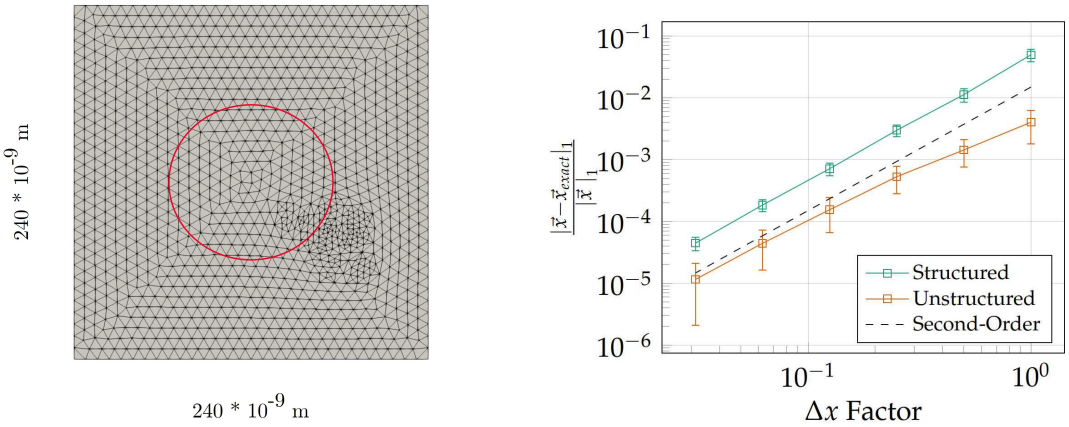
$$\omega^2 = \frac{q^2}{2\pi r^2 m\epsilon_0}. \tag{4.4}$$

Therefore we can define angular velocity $\omega = \sqrt{q^2/2\pi r^2 m\epsilon_0}$. As we know $x = r_{orbit}\cos(\omega t)$ and $y = r_{orbit}\sin(\omega t)$, it is now trivial to compare the simulated orbit to every point on the trajectory defined by the analytical solution.

The tests were carried out using a distorted unstructured mesh consisting of 2272 triangular elements in the base case. This mesh and its dimensions are shown in Fig. 4(a), with the base orbit trajectory shown in red. At the base level of refinement the mesh has an average Δx value of approximately 7×10^{-9} m, and a time-step size of $\Delta t \approx 1.846\times 10^{-10}$ s was used. This results in advection CFL ≈ 0.48. For this test we place the hydrogen ion at the centre of the mesh, directly on top of an element vertex. Data was collected for each refinement level using 100 randomized starting locations, varying the starting position by at most $\pm 0.5\times r_{orbit}$ m in each dimension. In this way we can determine the variation in the result due to the electron travelling through various levels of mesh distortion. The experiments were also carried out on a square structured mesh of quadrilateral elements at the same advection CFL value, with the base case consisting of 14 elements in both dimensions. Here we repeat the test placing the central particle at 100 randomized positions within the element quadrant. As a result of all cell quadrants being identical, we can obtain data that consider a representative range of possible particle positions within an element.

We now examine the effects of increasing levels of refinement on the L_1 error of the position of the orbiting electron against the analytical solution (normalised via the orbit radius), where we hold the ratio $\Delta x/\Delta t$ fixed in order to maintain a constant advection CFL value. At each subsequent level of refinement both Δx and Δt are halved.

Fig. 4(b) shows the results of this convergence study, with standard deviation error bars representing the change in the result due to the variation in the orbit location on the mesh. Note that the structured results start out with a higher error due to the low number of elements used in the base case. We can see that the standard deviation in the data is smoothly reduced as refinement level is increased for both structured and unstructured runs. This standard deviation is higher for the unstructured tests due the high level of mesh distortion, as expected. Additionally, we observe near the expected second-order convergence to the solution once the problem is sufficiently refined in the unstructured case. Exact second-order convergence is not achieved due to the high level of mesh distortion causing the result to be subject to larger errors. Conversely, the structured tests show the exact expected convergence throughout.



(a) Mesh used for unstructured orbit tests. The orbit base case is shown in red.

(b) Convergence study

Figure 4: A convergence study with fixed $\Delta t/\Delta x$ where (a) represents the geometry being studied and (b) shows the L_1 norm of the error in the position of the orbiting electron.

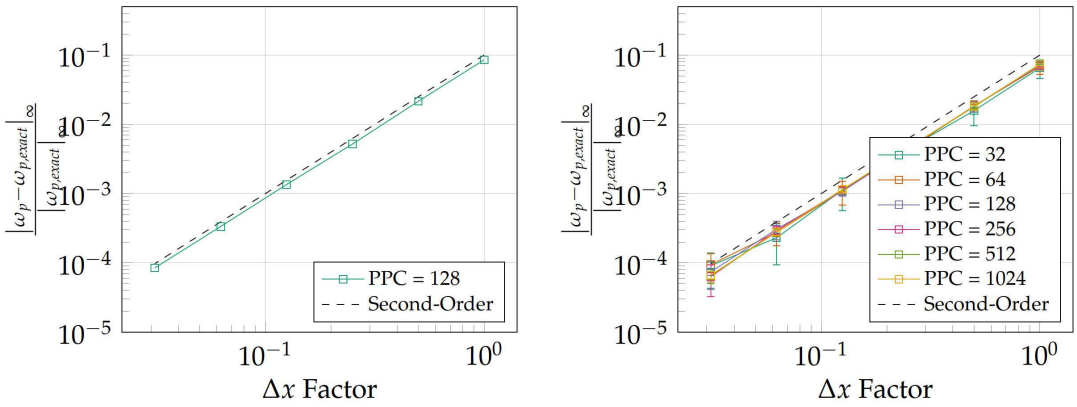
4.3 Electrostatic Langmuir wave

To determine the convergence behaviour of EMPIRE-PIC for electrostatic simulations we examine a cold one-dimensional Langmuir wave, a commonly-used benchmarking problem in the testing of PIC applications. The controlling parameters are as follows: we choose the number density $n_0 = 10^{14}$ m^{-3}, a temperature of 0 K, and we simulate a single plasma period. The problem domain is filled with a plasma consisting of electrons and hydrogen ions. Therefore, as the Langmuir wave is cold, we can derive the plasma frequency ω_p as below:

$$\omega_p = \sqrt{\frac{n_0 q^2}{m_e \epsilon_0}} \approx 564146027.6 \text{ rad/s}. \tag{4.5}$$

As EMPIRE-PIC is a 2D/3D code it cannot simulate a 1D problem directly; we instead set up a 2D mesh with periodic boundaries, and fixed $N_y = 2$ for all mesh refinement levels. For the base case of the convergence study we set $N_x = 16$, and an initial velocity perturbation of $v_x = 10^5$ m/s. Additionally, 16 time steps are used per period, resulting in advection CFL$\approx$0.0217. At each level of refinement both N_x and the number of time steps are doubled, maintaining a constant advection CFL value, while the initial perturbation is decreased by half. Finally, results are collected for a variety of particle per cell (PPC) counts, with one H^+ ion per cell, and the number of electrons per cell specified as an input parameter. As the simulation is refined the number of computational particles per cell is held constant.

We first consider results for a simulation using 128 particles per cell, uniformly loaded in the problem domain. This data is shown in Fig. 5(a), which plots the L_∞ norm of the



(a) Uniform particle layout

(b) Random-in-element particle layout

Figure 5: Plots showing results of convergence analysis performed on the Langmuir wave problem as the simulation is refined for constant $\Delta t/\Delta x$, demonstrating second-order convergence.

computed result at various refinement levels. In this case it is clear that we are achieving the second-order space and time convergence that we expect to see from the application. Extending this analysis, we also consider the convergence in the case where particles are randomly loaded within their respective elements. Fig. 5(b) shows the results of the same problem with 10 different initial particle loads, and various particle per cell counts. The variation in the result is represented by standard deviation error bars. We observe that increasing particle per cell count results in good noise reduction in the solution; refining the problem in space and time also has a beneficial effect. It is also evident that, as with the uniform particle layout, EMPIRE-PIC is achieving the theoretically expected convergence to the analytical solution.

4.4 Transverse electromagnetic wave in plasma

To test the convergence of EMPIRE-PIC for electromagnetic problems we consider an infinite, planar TEM wave propagating through an infinite neutral plasma. The solution to this problem is given in Chen [29], which derives the differences between a TEM wave in a vacuum versus in a plasma.

In this problem we choose controlling parameters as follows: The plasma number density is set as $n_0 = 10^{15}$ m^{-3}, with an electric field magnitude of $E_{mag} = 100$ V/m, and the vacuum frequency of the wave $f_v \approx 1.420$ GHz, and $\omega_v = 2\pi f_v$. We also assume that the electromagnetic wave is of such a high frequency that the hydrogen ions within the plasma remain stationary throughout the simulation, and that $\vec{J} \times \vec{B}$ forces are negligible. This has the effect that electrons are assumed to oscillate linearly in the plane of the electric field. From the above we can derive the plasma frequency and actual wave frequency

as follows:

$$\omega_p = \sqrt{\frac{n_0 q^2}{m_e \epsilon_0}} \approx 1.784 \times 10^9 \text{ rad/s}, \tag{4.6}$$

$$f = \frac{\omega}{2\pi} = \frac{1}{2\pi}\sqrt{\omega_p^2 + \omega_v^2} \approx 1.448 \text{ GHz}. \tag{4.7}$$

As the wave is infinite and steady, we can derive the constant phase velocity:

$$v_p = \sqrt{\frac{f}{f_v}} c \approx 1.02c > c. \tag{4.8}$$

This gives the maximum initial electron velocity as defined below, which is then initialized in phase with the electric field. The values of v_x and v_z are initialized to zero

$$v_y = \frac{qE_{mag}}{m_e \omega} \approx 1932.5 \text{ m/s}. \tag{4.9}$$

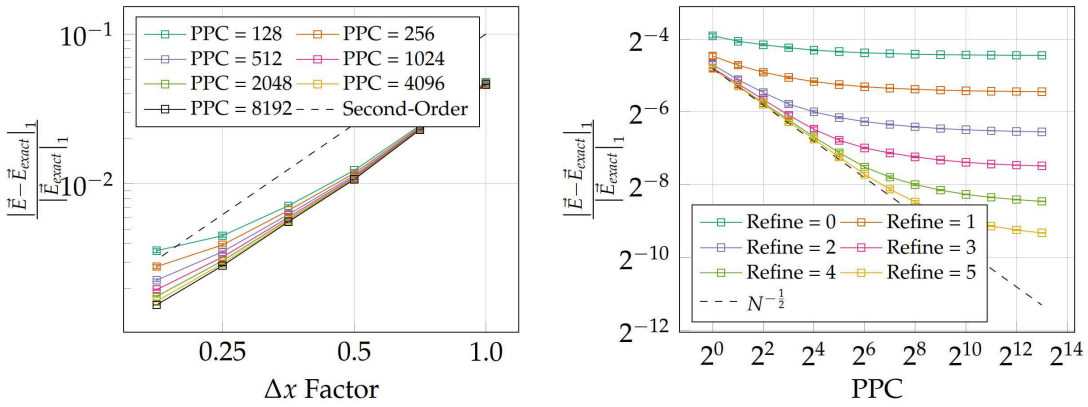
The velocity $\vec{u}$ of a given particle can now be calculated as follows, where p is the particle position projection onto the wavevector $(0,0,1)$:

$$\vec{u} = \vec{v} \times \sin\left(p + \frac{\pi}{2}\right) \text{ m/s}. \tag{4.10}$$

Finally, we define the maximum magnitude of the magnetic field such that it is congruous with the magnitude of the electric field

$$B_{mag} = \frac{\lambda}{2\pi}\frac{E_{mag}}{c^2}\left(\frac{n_0 q^2}{m_e \epsilon_0 \omega} + \omega\right) \approx 3.53 \times 10^{-7} \text{ T}. \tag{4.11}$$

From the derivation above it is simple to formulate a computational description of the problem. The problem is set up on a three-dimensional grid of hexahedral finite elements with periodic boundaries, effectively creating infinite space for the TEM wave. The wave is defined to travel in the z dimension of the computational mesh, with the majority of grid elements also in the z dimension – 12 in the base case. The x and y dimensions are each defined to have a constant 4 elements, and this is fixed for all simulation refinement levels. Additionally, the base case uses 50 simulation time steps resulting in a speed of light CFL ≈ 0.4707. As the ions are assumed to be stationary in the derivation, they are forced to remain immobile throughout the simulation. The computational particles are placed randomly within each element and weighted in order to achieve the specified plasma number density. We also ensure that the initial electron velocity is confined to the transverse direction in the plane of the electric field. The refinement scheme is as follows: at each refinement level both the number of time steps and the length of the domain is multiplied by a factor of $\sqrt{2}$, with the number of computational particles per element



(a) Convergence study using various PPC values.

(b) Particle count convergence for fixed refinements.

Figure 6: Plots showing results of the convergence study with fixed $\Delta t/\Delta x$ performed on the 3D TEM wave problem. The error in the electric field demonstrates second-order convergence to the analytical solution when sufficient numbers of particles are used.

held constant. Finally, we use the C-N time integration scheme discussed in Section 2.1.1 to advance the fields in this test.

Fig. 6 presents results for this problem collected at differing refinement levels and with various particle per cell counts. Each simulation is carried out with 10 different initial particle loads in order to examine the level of statistical noise in the solutions. Fig. 6(a) shows the results of a convergence study, where the L_1 norm of the electric field is the metric of interest. Additionally, Fig. 6(b) shows convergence with increased PPC counts for various fixed levels of refinement. In both cases, standard deviation error bars are used to represent the variation in error due to the noise introduced by the randomly loaded particles. It is clear to see that as the number of particles per cell is increased the convergence rate approaches the theoretical rates of second-order in both space and time that we expect from EMPIRE-PIC. We also observe that the level of noise in the solution decreases when the problem is refined or the number of particles per cell is increased. It is also clear that the code is achieving the expected convergence of $1/\sqrt{N_P}$ as PPC is increased for a given refinement level once the problem is refined enough in space/time for the improvements to be visible.

5 Performance of EMPIRE-PIC

In order to assess the performance, scalability, and portability of EMPIRE-PIC we present results collected from four distinct production HPC systems. The systems used in this paper are: Trinity, consisting of both Intel Xeon and Many Integrated Core (MIC) Intel Xeon Phi (KNL) partitions located at the Los Alamos National Laboratory; Astra, a Petascale

Table 4: Details of the systems used to collect EMPIRE-PIC performance data.

Machine	Nodes	Processor	Accelerator	Compiler(s)
Trinity (Haswell)	9436	2 × Intel Xeon E5-2698v3	-	Intel 18.0.5
Trinity (KNL)	9984	1 × Intel Xeon Phi 7250	-	Intel 18.0.5
Astra	2592	2 × Cavium Thunder-X2 CN9975	-	GCC 7.2.0
Sierra	4340	2 × IBM POWER9 22C	4 × NVIDIA V100	GCC 7.2.0 NVCC 9.2

Table 5: Details of problem sizes used to test EMPIRE-PIC.

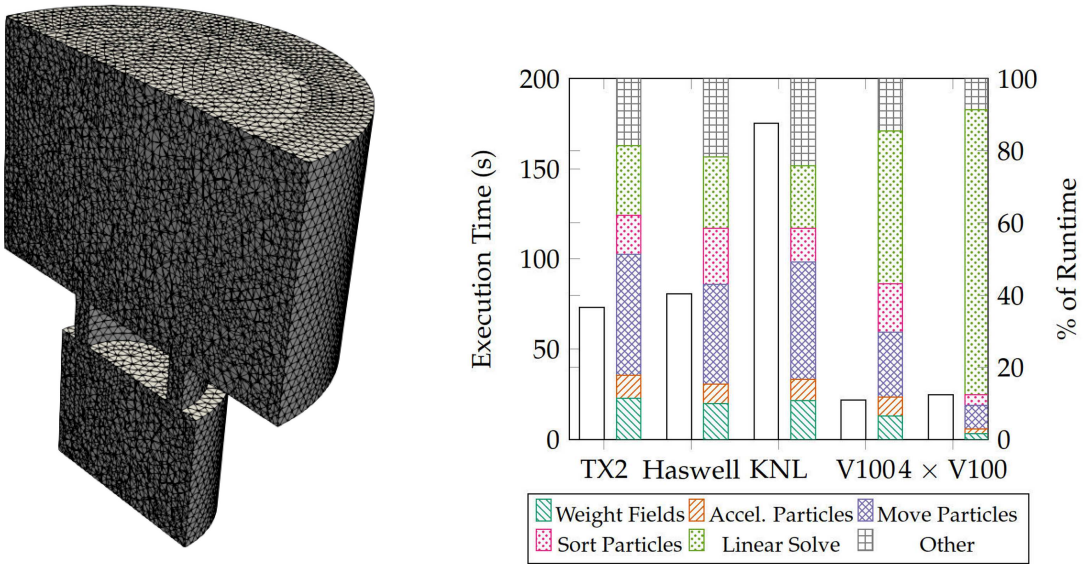
Size	# of Elements	# of Nodes	# of Edges	# of Faces	Particle Count	Particles/element
S	337.0 k	60.0 k	406.0 k	683.0 k	16.0 M	47.5
M	2.7 M	462.0 k	3.2 M	5.4 M	128.0 M	47.8
L	20.7 M	3.5 M	24.4 M	41.6 M	1.0 B	49.5
XL	166.0 M	27.9 M	195.0 M	333.0 M	8.2 B	49.4
XXL	1.3 B	223.0 M	1.6 B	2.7 B	65.6 B	49.2

ARM supercomputer based at SNL; and Sierra, an IBM POWER9 and NVIDIA V100 cluster installed at the LLNL. Specific details of these systems can be found in Table 4.

In this section, we use a three-dimensional electromagnetic problem on a tetrahedral mesh. A cross-section of this mesh is shown in Fig. 7(a). The problem domain is uniformly filled with equal amounts of both electrons and hydrogen ions to a number density of 1×10^{16} m^{-3}, with particles being loaded randomly within their assigned elements. The initial temperature is set to approximately 10 eV. The plasma is evolved for a time of 6×10^{-10} s over 100 simulation time-steps, and the CFL ranges from 3–30.

For the experiments on Trinity (Haswell) and Astra we adopted a hybrid approach using 1 MPI process per socket, with each process saturating all physical cores of the socket using OpenMP threads. For the KNL partition of Trinity, the nodes were used in quadrant mode with 1 MPI process per quadrant, and 16 OpenMP threads per process. Hyperthreading was not used for all CPU-based systems. For Sierra 1 MPI process is used per Tesla V100 GPU, resulting in 4 processes per node in total. A variety of problem sizes were tested on each system, with each problem size being approximately a factor of $8\times$ larger than the previous size in terms of both the mesh and the particle count. This facilitates both strong and weak scaling studies. Full details of all problem sizes are given in Table 5.

Fig. 7(b) shows the performance of EMPIRE-PIC at the single node level, running the small problem size across all systems. For the experiments on Sierra, runs were carried out using both a single V100 GPU, and all four V100 GPUs in the node. Along side the total execution time, Fig. 7(b) shows the proportion of the execution time spent in each of the kernels described in Section 3. As the problem is electromagnetic the cost of weighting the particles back to the mesh is included in the particle move time. The cost of migrating particles between processes is also included in the move kernel time. Finally,



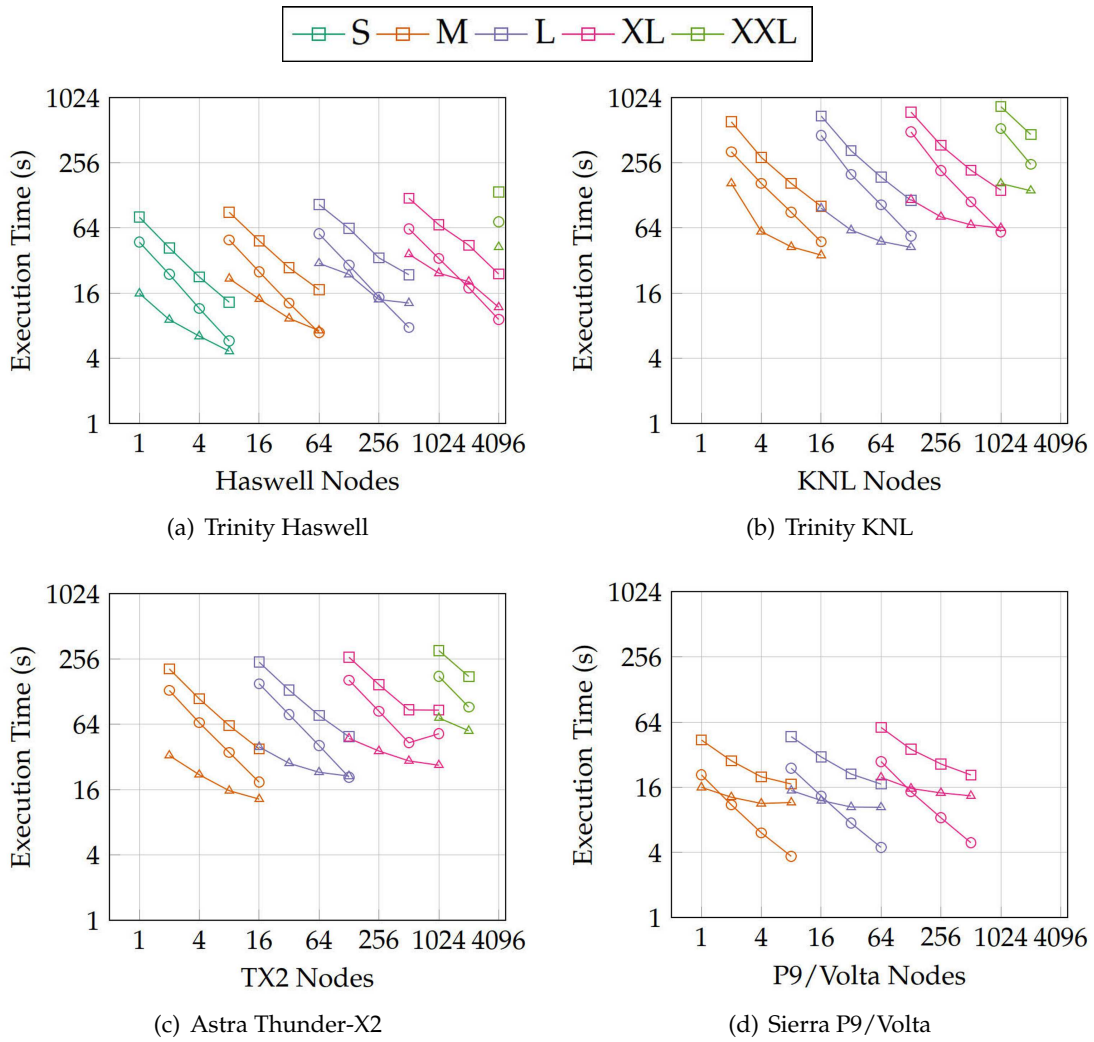
(a) (b)

Figure 7: Geometry used to test EMPIRE-PIC (left), and a breakdown of time spent in various application kernels for the small problem size (right).

the 'Other' time measurement includes time spent performing I/O, and other tasks such as filling views at the start of a time-step.

Our results show that the combined total time spent processing the particles is greater than for the linear solve of Maxwell's equations for all systems, with the exception Sierra when using all four available GPUs. We also observe comparable overall results when contrasting the performance of the ARM Thunder-X2 to the Haswell partition of Trinity, with Astra performing better on both the linear solve and the particle sort. This difference can be explained by the ARM system having double the number of memory channels of the Haswell system – a total of eight channels versus four providing an advantage for traditionally memory-bound algorithms. The traditional CPU systems also outperform the KNL partition of Trinity that makes use of Intel's Xeon Phi Knight's Landing MIC architecture. This performance disparity is most apparent when comparing the time taken to execute the particle move step.

When examining the performance data collected from Sierra it is clear to see that the Tesla V100 vastly outperforms CPU-based systems on particle based kernels. This is due to the massively parallel nature of GPU architecture combined with high-bandwidth memory allowing much greater numbers of particles to be processed simultaneously. This continues to hold true even when only a single Tesla V100 is used out of the four available on a single node. For the linear solver we observe similar performance on Sierra versus the three CPU systems when using a single GPU. When using all four available GPUs the linear solve on Sierra is slower than all other systems except the KNL. This is



(a) Trinity Haswell

(b) Trinity KNL

(c) Astra Thunder-X2

(d) Sierra P9/Volta

Figure 8: EMPIRE-PIC scaling study results for both partitions of Trinity, Astra, and Sierra. Squares, triangles, and circles represent the main time loop, linear solve, and particle kernels respectively.

caused by the linear solve strong scaling poorly across multiple GPUs when the problem size per GPU card is sufficiently small as the amount of work per GPU card becomes low.

Fig. 8 shows the results of both strong and weak scaling studies of EMPIRE-PIC for all of the aforementioned systems. We present data for the total time spent processing particles including MPI communications, the time spent solving for the updated fields, and the total execution time of the main loop of the application. It is clear to see that EMPIRE-PIC achieves near ideal strong scaling for the particle update on all systems even when high levels of strong scaling are applied. The linear solve strong scales well on both Haswell, KNL and Thunder-X2 but there is reduced strong scaling benefit observed

Table 6: Time taken to process a single particle or element, broken down by system. The crossover point is the amount of particles per cell needed to spend equal time processing particles and solving for the fields.

Machine	**μs / Particle**	**μs / Element**	**PPC Crossover Point**
Trinity (Haswell)	0.045	2.518	56.221
Trinity (KNL)	0.070	2.116	30.306
Astra	0.027	0.909	33.594
Sierra	0.003	0.219	83.867

for the V100 architecture. It is also evident that the solver quickly becomes the main performance bottleneck for the CUDA version of the code when sufficient levels of strong scaling are used.

With regards to weak scaling, EMPIRE-PIC scales well across all of the chosen systems demonstrating acceptable time-to-solution even as problem size is vastly increased versus the base case. Of particular note is that when running EMPIRE-PIC on Sierra we observe significantly reduced time-to-solution versus the three CPU-based systems while using a factor of eight fewer total compute nodes – this is unsurprising given the relative peak performance/memory bandwidth of the nodes in Sierra when compared to the other three systems.

Finally, Table 6 shows, for each machine, the time in microseconds spent to process a single particle and/or element, for the XL problem size after two levels of strong scaling have been applied. Using this data we can determine the number of particles per cell that would cause the amount of time spent processing the particles and solving for the updated fields to be equal. This is also denoted in Table 6 as the PPC crossover point, with the purpose of providing a view to the reader of how simulations not dominated by particle push time might perform on each system.

6 Conclusion

As we approach the major milestone of Exascale computing, modern computational architectures will continue to diversify. It is crucial that current and developing HPC applications can adapt to ever-increasing heterogeneity. This paper presents the development of EMPIRE-PIC, an unstructured electrostatic/electromagnetic FEM-PIC code capable of running simulations in both two- and three-dimensional spaces with realistic geometries. The code is written in C++ and uses the Kokkos programming model to achieve good performance from a single source across a variety of modern compute architectures, including Intel Xeon CPUs, Intel's Xeon Phi Knight's Landing, ARM Thunder-X2, and NVIDIA Tesla V100 GPUs attached to IBM POWER9 CPUs. To our knowledge, EMPIRE-PIC is the first unstructured electrostatic/electromagnetic FEM-PIC application to be developed using the Kokkos framework.

In this paper, we have demonstrated that both the electrostatic and electromagnetic

schemes implemented within EMPIRE-PIC achieve second-order accuracy in space and time via a convergence study using three benchmark problems: a simple electron orbit, an electrostatic Langmuir wave, and a transverse electromagnetic wave. In particular, the electrostatics scheme displays near-exact second-order convergence, and the electromagnetics scheme shows the correct second-order convergence once a sufficient number of particles are used.

We have also shown how such an FEM-PIC code can be implemented using the Kokkos and Trilinos libraries, demonstrating good performance across four distinct systems using a representative unstructured problem. Specifically we have observed near-ideal strong-scaling for the particle-based kernels. On the NVIDIA Tesla V100 platform we have seen relatively low levels of strong scalability for the linear solver, but good weak scaling is still achieved even up to problems consisting of more than one-hundred million mesh elements. It is also evident that the massively parallel nature of GPUs makes them well suited to the particle-based kernels in PIC codes, with Sierra outperforming the CPU based systems in almost all cases.

EMPIRE-PIC has been developed as part of Sandia's Advanced Simulation and Computing, Advanced Technology, Development, and Mitigation (ASC-ATDM) program to target next generation heterogeneous platforms. It will allow domain scientists to run complex simulations of plasma phenomena across a wide range of the Department of Energy's future Exascale supercomputers at a scale far greater than existing production codes.

6.1 Future work

There are a number of opportunities to explore improvements to both the accuracy and the performance of EMPIRE-PIC. Firstly, the use of higher-order particle representations has been explored within EMPIRE-PIC. Specifically, work is underway to evaluate novel implementations of higher-order particle shape functions in the context of unstructured FEM-PIC [26, 27].

Finally, PIC codes also experience performance degradation during long simulations due to increased load imbalance as a result of the dynamic nature of the particle-based parts of the simulation workload. One approach to resolve this issue is the use of Asynchronous Many Tasking (AMT) libraries, enabling the use of dynamic load balancing techniques to alleviate the performance penalty of an unbalanced simulation. The use of AMT libraries within EMPIRE-PIC is currently being explored.

Acknowledgments

This work was supported by the UK Atomic Weapons Establishment (AWE) under grant CDK0724 (AWE Technical Outreach Programme). Professor Stephen Jarvis is an AWE William Penney Fellow.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] Tempus Trilinos package. `https://github.com/trilinos/Trilinos/tree/master/packages/tempus`.
[2] OpenCL specification version 2.0. Technical report, Jul 2015. `https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf`.
[3] CUDA C++ programming guide. Technical report, NVIDIA, Nov 2019. `https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf`.
[4] oneAPI programming model. Technical report, Intel Corporation, Nov 2019. `https://www.oneapi.com/`.
[5] Top500, 2020. [Online; accessed 30-June-2020].
[6] M F Adams, S Ethier, and N Wichmann. Performance of particle in cell methods on highly concurrent computational architectures. In *Journal of Physics: Conference Series*, volume 78, page 012001. IOP Publishing, 2007.
[7] T D Arber, K Bennett, C S Brady, A Lawrence-Douglas, M G Ramsay, N J Sircombe, P Gillies, R G Evans, H Schmitz, A R Bell, and C P Ridgers. Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Physics and Controlled Fusion*, 57(11):113001, 2015.
[8] Eric Bavier, Mark Hoemmen, Sivasankaran Rajamanickam, and Heidi Thornquist. Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems. *Scientific Programming*, 20(3):241–255, July 2012.
[9] Donald J Becker, Thomas Sterling, Daniel Savarese, John E Dorband, Udaya A Ranawak, and Charles V Packer. BEOWULF: A Parallel Workstation for Scientific Computation. In *Proceedings of the International Conference on Parallel Processing (ICPP'95)*, pages 11–14, 1995.
[10] Luc Berger-Vergiat, Christian A. Glusa, Jonathan J. Hu, Matthias Mayr, Andrey Prokopenko, Christopher M. Siefert, Raymond S. Tuminaro, and Tobias A. Wiesner. MueLu user's guide. Technical Report SAND2019-0537, Sandia National Laboratories, 2019.
[11] R F Bird, S J Pennycook, S A Wright, and S A Jarvis. Towards a portable and future-proof particle-in-cell plasma physics code. In *1st International Workshop on OpenCL (IWOCL 13)*, May 2013.
[12] Robert F Bird, Patrick Gillies, Michael R Bareford, John Andy Herdman, and Stephen A. Jarvis. Performance optimisation of inertial confinement fusion codes using mini-applications. *The International Journal of High Performance Computing Applications*, 32(4):570–581, 2018.
[13] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation*. Plasma Physics Series. Institute of Physics Publishing, Bristol BS1 6BE, UK, 1991.
[14] Charles K Birdsall and Dieter Fuss. Clouds-in-clouds, clouds-in-cells physics for many-body plasma simulation. *Journal of Computational Physics*, 3(4):494 – 511, 1969.

[15] Joseph D Blahovec, Lester A Bowers, John W Luginsland, Gerald E Sasser, and John J Watrous. 3-d icepic simulations of the relativistic klystron oscillator. *IEEE transactions on plasma science*, 28(3):821–829, 2000.

[16] OpenMP Architecture Review Board. OpenMP application programming interface version 5.0. Technical report, Nov 2018. `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf`.

[17] P. Bochev, H. C. Edwards, R. C. Kirby, K. Peterson, and D. Ridzal. Solving PDEs with Intrepid. *Scientific Programming*, 20(2):151–180, 2012.

[18] P. Bochev and J. Hyman. Principles of mimetic discretizations of differential operators. *Compatible spatial discretizations*, pages 89–119, 2006.

[19] Pavel B Bochev, Jonathan J Hu, Christopher M Siefert, and Raymond S Tuminaro. An algebraic multigrid approach based on a compatible gauge reformulation of Maxwell's equations. *SIAM Journal on Scientific Computing*, 31(1):557–583, 2008.

[20] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering, and Coloring. *Scientific Programming*, 20(2):129–150, 2012.

[21] J Boris. Relativistic Plasma Simulation: Optimization of a Hybrid Code. In *Proceedings of the Fourth Conference on Numerical Simulation of Plasmas*, pages 3–68, Naval Research Laboratory, Washington, D.C, July 1971.

[22] A. Bossavit. A rationale for 'edge-elements' in 3-d fields computations. *IEEE Transactions on Magnetics*, 24(1):74–79, 1988.

[23] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 pflop/s trillion-particle kinetic modeling of laser plasma interaction on roadrunner. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 63:1–63:11, Piscataway, NJ, USA, 2008. IEEE Press.

[24] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. *Physics of Plasmas*, 15(5):055703, 2008.

[25] Franco Brezzi and Michel Fortin. *Mixed and hybrid finite element methods*, volume 15. Springer Science & Business Media, 2012.

[26] Dominic A.S. Brown, Matthew T. Bettencourt, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. Higher-Order Particle Representation for Particle-in-Cell Simulations. *Journal of Computational Physics*, 435:110255, June 2021.

[27] Dominic A.S. Brown, Steven A. Wright, and Stephen A. Jarvis. Performance of a Second Order Electrostatic Particle-in-Cell Algorithm on Modern Many-Core Architectures. *Electronic Notes in Theoretical Computer Science*, 340:67–84, October 2018.

[28] Heiko Burau, Renée Widera, Wolfgang Honig, Guido Juckeland, Alexander Debus, Thomas Kluge, Ulrich Schramm, Tomas E Cowan, Roland Sauerbrey, and Michael Bussmann. PIConGPU: a fully relativistic particle-in-cell code for a GPU cluster. *IEEE Transactions on Plasma Science*, 38(10):2831–2839, 2010.

[29] Francis F Chen. *Introduction to Plasma Physics and Controlled Fusion*. Springer, third edition, 2016.

[30] G. Chen, L. Chacón, and D.C. Barnes. An efficient mixed-precision, hybrid CPU-GPU implementation of a nonlinearly implicit one-dimensional particle-in-cell algorithm. *Journal of Computational Physics*, 231(16):5374–5388, 2012.

[31] OpenACC Language Committee et al. OpenACC application programming interface version 2.6. Technical report, Nov 2017. `https://www.openacc.org/sites/default/files/`

inline-files/OpenACC.2.6.final.pdf.
[32] John Dawson. Onedimensional plasma model. *The Physics of Fluids*, 5(4):445–459, 1962.
[33] John M. Dawson. Particle Simulation of Plasmas. *Reviews of Modern Physics*, 55:403–447, Apr 1983.
[34] Viktor K. Decyk and Tajendra V. Singh. Adaptable particle-in-cell algorithms for graphical processing units. *Computer Physics Communications*, 182(3):641–648, 2011.
[35] Thomas H Dupree. Kinetic Theory of Plasma and the Electromagnetic Field. *Physics of Fluids (1958-1988)*, 6(12):1714–1729, 1963.
[36] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling Manycore Performance Portability Through Polymorphic Memory Access Patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.
[37] 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, and J. C. Adam. OSIRIS: A three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators. In Peter M. A. Sloot, Alfons G. Hoekstra, C. J. Kenneth Tan, and Jack J. Dongarra, editors, *Computational Science ICCS 2002*, pages 342–351, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
[38] Gregory Fridman, Gary Friedman, Alexander Gutsol, Anatoly B. Shekhter, Victor N. Vasilets, and Alexander Fridman. Applied plasma medicine. *Plasma Processes and Polymers*, 5(6):503–533, 2008.
[39] Alex Friedman. A second-order implicit particle mover with adjustable damping. *Journal of Computational Physics*, 90(2):292–312, October 1990.
[40] Kai Germaschewski, William Fox, Stephen Abbott, Narges Ahmadi, Kristofor Maynard, Liang Wang, Hartmut Ruhl, and Amitava Bhattacharjee. The plasma simulation code: A modern particle-in-cell code with patch-based load-balancing. *Journal of Computational Physics*, 318:305–326, 2016.
[41] Philip M Gresho and Robert L Lee. Don't suppress the wiggles—they're telling you something! *Computers & Fluids*, 9(2):223–253, June 1981.
[42] Khronos OpenCL Working Group et al. SYCL specification version 1.2.1. Technical report, Nov 2019. https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf.
[43] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An Overview of the Trilinos Project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.
[44] R. Hiptmair. Finite elements in computational electromagnetism. *Acta Numerica*, 11:237–339, 2002.
[45] R. W. Hockney. Computer experiment of anomalous diffusion. *The Physics of Fluids*, 9(9):1826–1835, 1966.
[46] Roger W Hockney and James W Eastwood. *Computer simulation using particles*. crc Press, 1988.
[47] Rich Hornung, Holger Jones, Jeff Keasler, Rob Neely, Olga Pearce, Si Hammond, Christian Trott, Paul Lin, Courtenay Vaughan, Jeanine Cook, Rob Hoekstra, Ben Bergen, Josh Payne, and Geoff Womeldorff. ASC tri-lab co-design level 2 milestone report 2015. Technical report, Lawrence Livermore National Laboratory, Sep 2015.
[48] Jianming Jin. *The Finite Element Method in Electromagnetics*. Wiley-IEEE Press, 3rd edition, 2014.

[49] Y. L. Klimontovich. *The Statistical Theory of Non-Equilibrium Processes in a Plasma: International Series of Monographs in Natural Philosophy*, volume 9. Elsevier, 2013.
[50] S. Ku, R. Hager, C.S. Chang, J.M. Kwon, and S.E. Parker. A new hybrid-lagrangian numerical scheme for gyrokinetic simulation of tokamak edge plasma. *Journal of Computational Physics*, 315:467 – 475, 2016.
[51] A. B. Langdon and C. K. Birdsall. Theory of plasma simulation using finitesize particles. *The Physics of Fluids*, 13(8):2115–2122, 1970.
[52] A Bruce Langdon. On enforcing gauss' law in electromagnetic particle-in-cell codes. *Computer Physics Communications*, 70(3):447–450, 1992.
[53] R. Marchand. Ptetra, a tool to simulate low orbit satellite–plasma interaction. *IEEE Transactions on Plasma Science*, 40(2):217–229, Feb 2012.
[54] Barry Marder. A method for incorporating gauss' law into electromagnetic pic codes. *Journal of Computational Physics*, 68(1):48–55, 1987.
[55] Collin S Meierbachtol, Andrew D Greenwood, John P Verboncoeur, and Balasubramaniam Shanker. Conformal electromagnetic particle in cell: A review. *IEEE Transactions on Plasma Science*, 43(11):3778–3793, 2015.
[56] Sean T Miller, Eric C Cyr, John N Shadid, Richard Michael Jack Kramer, Edward Geoffrey Phillips, Sidafa Conde, and Roger P Pawlowski. IMEX and exact sequence discretization of the multi-fluid plasma model. *Journal of Computational Physics*, 397:108806, 2019.
[57] Jean-Claude Nédélec. Mixed finite elements in $\mathbb{R}^3$. *Numerische Mathematik*, 35(3):315–341, 1980.
[58] D. R. Nicholson. *Introduction to Plasma Theory*. Krieger Publishing Company, Malabar, Florida, 1992.
[59] S. J. Pennycook, J. D. Sewall, and V. W. Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, pages 947–958, 2017.
[60] Martin Campos Pinto, Marie Mounier, and Eric Sonnendrücker. Handling the divergence constraints in maxwell and vlasov–maxwell simulations. *Applied Mathematics and Computation*, 272:403–419, 2016.
[61] Timothy D Pointon. Second-order, exact charge conservation for electromagnetic particle-in-cell simulation in complex geometry. *Computer Physics Communications*, 179(8):535–544, 2008.
[62] F. Rapetti and A. Bossavit. Whitney forms of higher degree. *SIAM Journal on Numerical Analysis*, 47(3):2369–2386, 2009.
[63] Mario A. Riquelme, Eliot Quataert, and Daniel Verscharen. Particle-in-cell simulations of continuously driven mirror and ion cyclotron instabilities in high beta astrophysical and heliospheric plasmas. *The Astrophysical Journal*, 800(1):27, Feb 2015.
[64] J. Roussel, F. Rogier, G. Dufour, J. Mateo-Velez, J. Forest, A. Hilgers, D. Rodgers, L. Girard, and D. Payan. Spis open-source code: Methods, capabilities, achievements, and prospects. *IEEE Transactions on Plasma Science*, 36(5):2360–2368, Oct 2008.
[65] Aaron Scheinberg, Guangye Chen, Stephane Ethier, Stuart Slattery, Robert F. Bird, Pat Worley, and Choong-Seock Chang. Kokkos and fortran in the exascale computing project plasma physics code xgc. 2019.
[66] Stuart Slattery, Christoph Junghans, Damien Lebrun-Grandie, Shane Fogerty, Robert F. Bird, Sam Reeve, Guangye Chen, Rene Halver, Aaron Scheinberg, Cameron Smith, and Evan Weinberg. Ecp-copa/cabana: Version 0.3.0, May 2020.
[67] W. Tang, B. Wang, S. Ethier, G. Kwasniewski, T. Hoefler, K. Z. Ibrahim, K. Madduri, S. Williams, L. Oliker, C. Rosales-Fernandez, and T. Williams. Extreme scale plasma tur-

bulence simulations on top supercomputers worldwide. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 502–513, Nov 2016.

[68] J.-L. Vay, A. Almgren, J. Bell, L. Ge, D. P. Grote, M. Hogan, O. Kononenko, R. Lehe, A. Myers, C. Ng, J. Park, R. Ryne, O. Shapoval, M. Thévenet, and W. Zhang. 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*, 909:476–479, 2018.

[69] J.-L. Vay, P. Colella, J. W. Kwan, P. McCorquodale, D. B. Serafini, A. Friedman, D. P. Grote, G. Westenskow, J.-C. Adam, A. Héron, and I. Haber. Application of adaptive mesh refinement to particle-in-cell simulations of plasmas and beams. *Physics of Plasmas*, 11(5):2928–2934, 2004.

[70] John Villasenor and Oscar Buneman. Rigorous charge conservation for local electromagnetic field solvers. *Computer Physics Communications*, 69(2):306–316, 1992.

[71] B. Wang, S. Ethier, W. Tang, T. Williams, K. Z. Ibrahim, K. Madduri, S. Williams, and L. Oliker. Kinetic turbulence simulations at extreme scale on leadership-class systems. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2013.

[72] E. Wang, S. Wu, Q. Zhang, J. Liu, W. Zhang, Z. Lin, Y. Lu, Y. Du, and X. Zhu. The Gyrokinetic Particle Simulation of Fusion Plasmas on Tianhe-2 Supercomputer. In *2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 25–32, Nov 2016.

[73] Kane S. Yee. Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media. *IEEE Transactions on Antennas and Propagation*, pages 302–307, 1966.