


T/AW087/21
Support and Coordination

Report 2057699-TN-03-02 

Progress on Development of an FEM-PIC Miniapp

Steven Wright and Edward Higgins

University of York

Gihan Mudalige, Ben McMillan, and Tom Goffrey

University of Warwick

February 9, 2023

Changelog

January 2023

- Made all corrections from Nov 2022 (see attached ‘Corrections’ document).
- Included an updated performance section to represent the current state of the application’s performance.

September 2022

- Updated design and implementation section to include details of the mini-app developed so far.
- Added some initial performance results to be iterated on later.

1 Executive Summary

The goal of the NEPTUNE project (**NE**utrals & **P**lasma **TU**rbulence **N**umerics for the **E**xascale) is the development a new code for the simulation of a next generation fusion reactor. The initial focus of NEPTUNE is in the simulation of the edge region of a tokamak and the “exhaust”, or “divertor”, region. Modelling this edge region likely requires the use of both fluid-based and particle-based models working in tandem.

Consequently, Project NEPTUNE is based around four work streams, where **FM-WP2** (plasma multiphysics model) is focused on fluid models and **FM-WP3** (neutral gas & impurity model) is focused on particle-based methods. Some of the work in these packages has been based around Bout++, Nektar++ and EPOCH. Bout++ and Nektar++ are both fluid models based on finite differencing and a spectral element method, respectively, while EPOCH is a particle-in-cell (PIC) application.

This work package (**FM-WP4** (code structure and coordination)) is in concerned with establishing “best practices” when engineering of a simulation code combining elements of these applications with a focus on performance and portability, in particular to heterogeneous Exascale-class architectures.

In our previous project (T/NA086/20) we evaluated a number of mini-applications in order to assess the performance portability of various approaches to developing Exascale-ready software. We evaluated a number of codes that implement a fluid model, using a variety of methodologies, and did so in a number of programming models. This has provided valuable insight into the performance portability of various approaches to heterogeneous software development for fluid-based applications.

However, our initial investigation of particle-methods was based on a limited set of applications, with each only implemented in the Kokkos performance portability layer. While there are a number of PIC codes available for evaluation (e.g. EPOCH, PICSAR, WarpX, EMPIRE-PIC), these are typically very large codes, implemented in only a single programming language. Assessing performance portable approaches to programming PIC codes is therefore difficult with these codebases.

This work package therefore seeks to develop a new PIC mini-application that is small enough that it can be used for evaluation, but complex enough to be somewhat representative of the PIC element of NEPTUNE – in particular the particle pushing kernels. Since NEPTUNE will likely require the use of complex geometries, our PIC code will be based on an unstructured grid approach, and thus will necessitate indirect memory access patterns.

This report documents our progress towards the development of this mini-application. The mini-application is available for evaluation here: <https://github.com/ejh516/mini-pic>¹

¹We will move this repository to the ExCALIBUR-NEPTUNE github organisation when licensing issues are resolved. It will then move to: <https://github.com/ExCALIBUR-NEPTUNE/mini-fem-pic>

2 Requirements

The mini-application that will be developed as part of this work package will be based on the Particle-in-Cell (PIC) method, using a Finite-Element Method (FEM) field solver. This is similar to the EMPIRE-PIC application [1], but miniaturised to allow rapid evaluation of performance portable programming models.

2.1 The Particle-in-Cell Method

The PIC method is a well established procedure for modelling the behaviour of charged particles in the presence of electric and magnetic fields [2, 3]. Discrete particles are tracked in a Lagrangian frame, while the electric and magnetic fields are stored on stationary points on a fixed Eulerian mesh.

The electric and magnetic fields evolve according to Maxwell's equations (Equations (1)-(4)).

$$\nabla \cdot \vec{E} = \frac{\rho}{\epsilon_0} \tag{1}$$

$$\nabla \cdot \vec{B} = 0 \tag{2}$$

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} \tag{3}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\mu_0 \epsilon_0} \nabla \times \vec{B} - \frac{1}{\epsilon_0} \vec{J} \tag{4}$$

While the force experienced by a particle is calculated according to the Lorentz force (Equation (5)).

$$\vec{F} = q \left(\vec{E} + \vec{v} \times \vec{B} \right) \tag{5}$$

A typical PIC method can be thought of as two coupled solvers where one is responsible for updating the electric and magnetic fields according the Maxwell's equations, while another calculates the movement of particles according to the Lorentz force. These are referred to as the *field solver* and the *particle mover* (sometimes called the *particle pusher*), respectively.

The main time loop of the core PIC algorithm consists of: solving 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 back to the mesh. The algorithm is summarised in Figure 1.

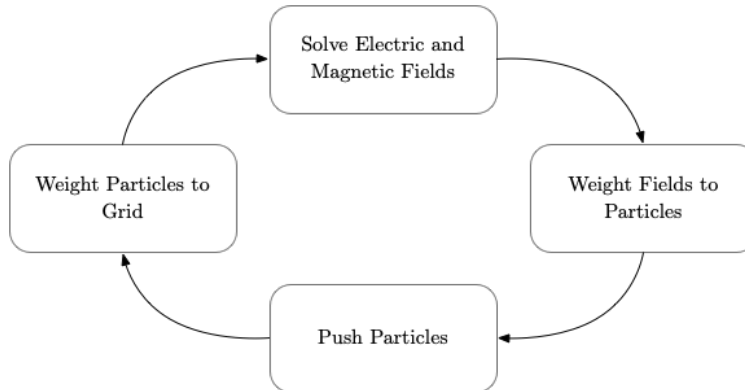


Figure 1: Flow chart summarising the key components of the PIC algorithm

2.2 The Finite Element Method

Many PIC codes rely on the finite-difference time-domain (FDTD) method devised by Yee in 1966 [4]. In these codes, the time-dependent PDEs are solved using a fixed, staggered grid in a leapfrog manner. EPOCH and VPIC are both examples of this, where particles are tracked on a structured square or cubic grid [5, 6].

For complex geometries (i.e. angled and curved) a regular structured grid approach requires a mesh with high spatial resolution and significant approximation, leading to increased computational intensity and a loss of accuracy. Instead, for these geometries alternative methods may be required. Bout++ uses a curvilinear coordinate system to more accurately represent curved geometries [7], while Nektar++ can use a high-order finite-element scheme with spectral element basis functions to more closely fit a mesh to complex geometries [8].

In the standard PIC method, before calculating the movement of particles, the value of the electric and magnetic fields must be calculated at each particle position. For the structured case, this may be a simple interpolation based on the distance from each edge of the containing cell. For the unstructured case, this may represent a significant computational undertaking.

One approach to updating the electric and magnetic fields for an unstructured grid is the *finite element method*. This involves splitting the simulation region up into discrete cells and approximating the values of the fields from the values of the fields at particular points within these cells. There are a number of different methods for discretising space into finite elements and storing the field values on element edges and faces.

The EMPIRE-PIC code, developed at Sandia National Laboratories, is an example of an FEM-PIC code, solving the electric and magnetic fields on a tetrahedral mesh [1]. In EMPIRE-PIC, the particle kernels (weighting of the fields to particles, the weighting of charge/current to the grid and the movement of particles) represents a greater proportion of the execution time than the linear solve on most platforms. Evaluating particle-based methods on a structured code is likely to partially obfuscate the computational cost of these particle kernels.

In order to provide a more realistic view of the likely performance and performance portability of approaches to these particle methods therefore requires a more representative mini-application.

2.3 Requirements for an FEM-PIC mini-application

Our previous evaluation of approaches to performance portability was primarily concerned with fluid-based simulations, due to a lack of available PIC mini-applications. The PIC applications that were evaluated were only available in a single programming model each, and in some cases were not representative of the NEPTUNE use-case.

Therefore the remainder of this report details the development of a simple FEM-PIC mini-application to facilitate the exploration of performance portable approaches to Exascale software development for unstructured-mesh, particle-based codes.

The guiding principles for this mini-application are:

1. A simple, understandable implementation of the a basic FEM solver and appropriate particle kernels.
2. At most 5,000 lines of C/C++ code (and preferably less than 3,000), to facilitate rapid redeployment.
3. Concise and appropriate diagnostic information, with timing/performance information relating to each of the key particle kernels.

3 Design and Implementation

In the previous iteration of this report (2057699-TN-03-01), we reviewed a number of potential base applications for the mini-application. We have since discovered an existing PIC code written in C++ that is relatively simple (fewer than 1500 lines of code) and uses unstructured grids, as is required for this project [9]. This will serve as the base for our mini-application.

The base application implements the electrostatic PIC method (i.e. it assumes that $\frac{\partial \vec{B}}{\partial t} = 0$). Since the inclusion of magnetic fields adds complexity to the codebase without changing the performance profile significantly, we shall continue in the electrostatic regime for our mini-application.

The base application is only set up to simulate a single system: ion flow past a charged sphere. As such, the code will be modified to allow a range of systems to be studied. This will allow us to explore algorithm performance across a range of system sizes and configurations.

Figure 2 provides an overview of the mini-application, demonstrating the three key kernels identified in the previous iteration of this report.

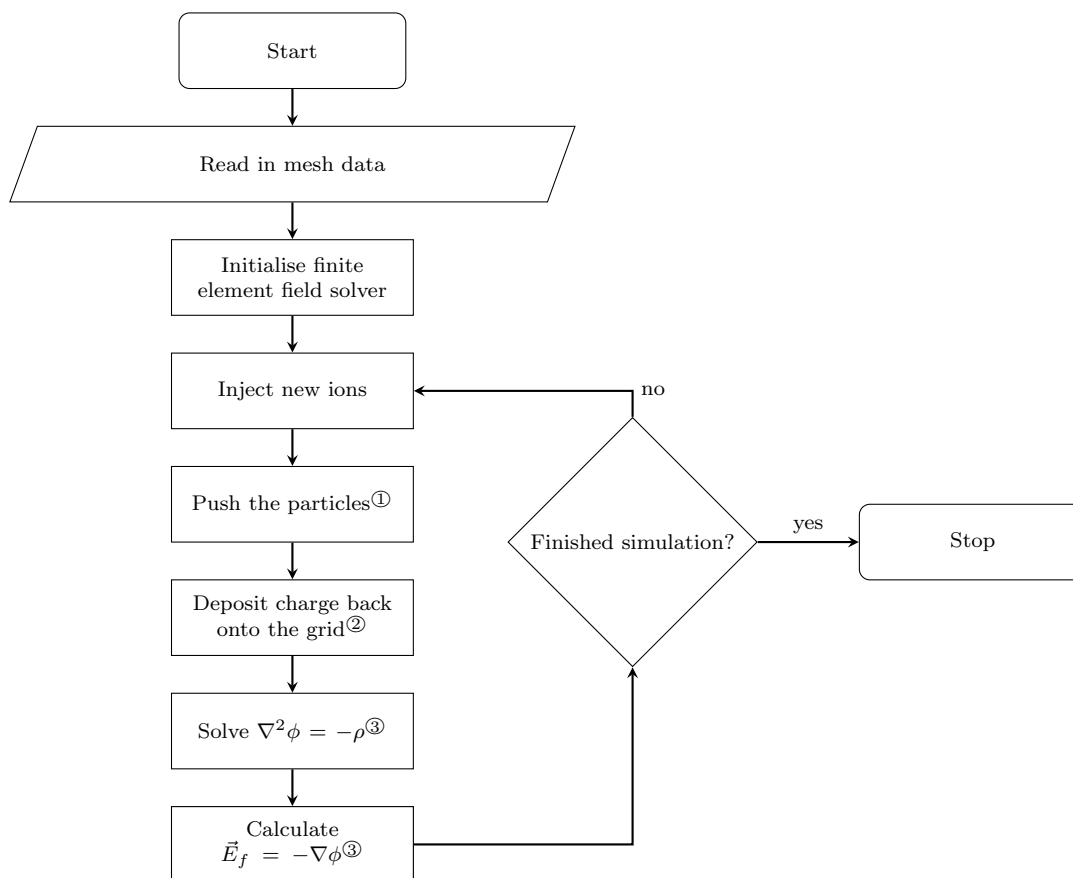


Figure 2: Program flow of the mini-application, identifying the three key computational kernels: 1) particle pushing, 2) depositing charge and 3) calculating fields.

3.1 Computational Kernels

3.1.1 Pushing Particles

In the previous iteration of this report, there were discussions around how the particles were stored in memory. For this mini-application, the particles are currently stored as an Array-of-structs (AoS), as shown in Figure 3. As was previously discussed, this is likely better than storing the particles in a linked list, as

```
1 /*particle class*/
2 struct particle {
3     double pos[3];
4     double vel[3];
5     double lc[4];      /*particle's weights*/
6     int cell_index;   /*last cell known to contain this particle*/
7 };
8
9 /*species class*/
10 class Species {
11 public:
12     std::vector<particle> particles;
13     double mass;
14     double charge;
15     // etc...
16 };
```

Figure 3: The structure of a species of particles in the mini-application.

is the case in EPOCH [5]. It is not entirely clear whether it is better using structs-of-arrays or arrays-of-structs [10], or whether particles should be stored in a global list, or in a list per-cell (and indeed whether different approaches suit different hardware). These are considerations we can rapidly investigate using this mini-application.

Since we are neglecting the magnetic field, a full electromagnetic integrator that takes into account the magnetic field, such as the Boris method [11], is not required. Instead a simpler leapfrog algorithm should be sufficient [12]. This works by storing the positions and velocities offset from one another by half a time step, as described below:

$$\vec{v}_{-\frac{1}{2}} = \vec{v}_0 - \frac{q\vec{E}_0}{2m}\Delta t \quad (6)$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_{n+\frac{1}{2}}\Delta t \quad (7)$$

$$\vec{v}_{n+\frac{1}{2}} = \vec{v}_{n-\frac{1}{2}} + \frac{q\vec{E}_n}{m}\Delta t \quad (8)$$

While higher order time integration algorithms do exist, it is not necessary to investigate them in this project. This is because for the full electrodynamic case that is of interest, the Boris method is known to work well.

3.1.2 Depositing Charge

Once the particles have been moved, the charge density needs to be updated. This is done by first finding which cell each particle is in, and then distributing that particle across the nodes of that cell.

In order to calculate which cell an existing particle is in, a neighbour search is performed. This is done by first looking in the cell that the particle was last in, then searching recursively its neighbours in the direction indicated by the cell functions. The process for doing this is described in reference [9].

New particles are created with random positions on a set of user-specified “inlet” faces. In this case, the cell that the particle is in is already known, so there is no cost involved with calculating it.

3.1.3 Calculating Fields

When the charge density has been evaluated, the ϕ and \vec{E} fields can be calculated. This is done using the finite element method (FEM) [13]. Fundamentally, ϕ needs to be calculated by solving the linear Poisson equation:

$$\nabla^2\phi = -\rho \tag{9}$$

This can be reformulated to the matrix equation:

$$\mathbf{K}\mathbf{d} = \mathbf{F} \tag{10}$$

where \mathbf{K} is the stiffness matrix which contains information about the finite element discretisation, \mathbf{d} is the solution vector which relates to the values of ϕ on the nodes and \mathbf{F} is the force vector whose elements contain the negative of the inner product of the density with the basis functions.

The vector \mathbf{d} can be calculated by solving Equation (10). In the mini-app, this equation may be solved using one of two methods. This code implements both a simple Gauss-Seidel linear solver and an interface to the LAPACK library routine `DGSEV`². The LAPACK routine has the advantage that, for a sufficiently optimised library, the performance should be near-optimal for any particular architecture. It also often has features like multi-threading available “out of the box”. However if no LAPACK library is available, or if an un-optimised library is provided, the Gauss-Seidel function is able to provide the same functionality. Having the routine explicitly coded in the program also allows for manual optimisation and parallelisation, targeting specific hardware.

The electric field \vec{E} can now be calculated by finding the gradient of ϕ :

$$\vec{E} = -\nabla\phi \tag{11}$$

²Since the Jacobian matrix should be symmetrical, the `DSYSV` routine would likely be more efficient, and require only half of the Jacobian to be required. This will be investigated further in the future.

3.2 Performance Profiling

In order to assess the performance of the mini-application, a simple function tracer/profiler was added, inspired by timing routines taken from the CASTEP [14] materials modelling code. This works by recording the time that each function is entered or exited, and aggregating the times over the function names. The results of this aggregation are stored in an object that can later be queried to get profiling information.

As each function of interest is called, a `TraceCaller` object is created and the constructor records the current wall time. At the end of the function, as this object leaves scope, its destructor records the time again and records the time elapsed between the two. These operations call `enter()` and `exit()` methods on the object that holds the profiling information. Figure 4 demonstrates how a function can be traced and the results stored in the `trace::current` object.

```
1 #define TRACE_ME TraceCaller _TRACE_OBJECT(__func__);
2
3 TraceCaller(std::string name_) : name(std::move(name_)) {
4     trace::current.enter(name);
5 }
6
7 ~TraceCaller() {
8     trace::current.exit(name);
9 }
10
11 void someInterestingFunc { TRACE_ME;
12     ...
13 }
```

Figure 4: An example of how a function’s performance is profiled.

When considering parallel programs, it is important to think about how this will behave. In terms of traditional MPI workloads, each process can generate its own profile which the user can aggregate themselves after the application has run. Similarly, libraries such as OpenMP do not have a problem as routines are generally entered together. In this case sensible decisions can be made, such as recording the time only on the master thread. When it comes to more complicated parallel workflows however, some thought will have to go into how to record and aggregate this information.

4 Performance

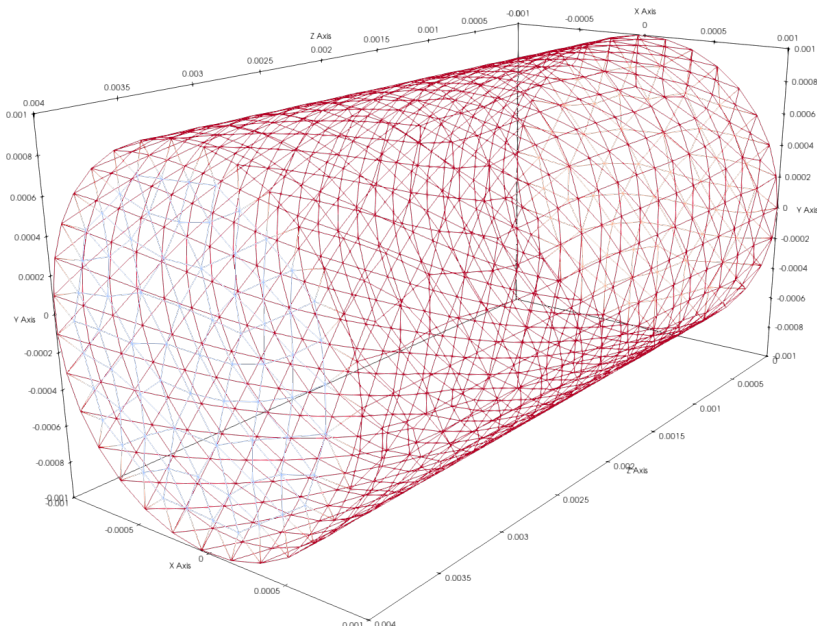


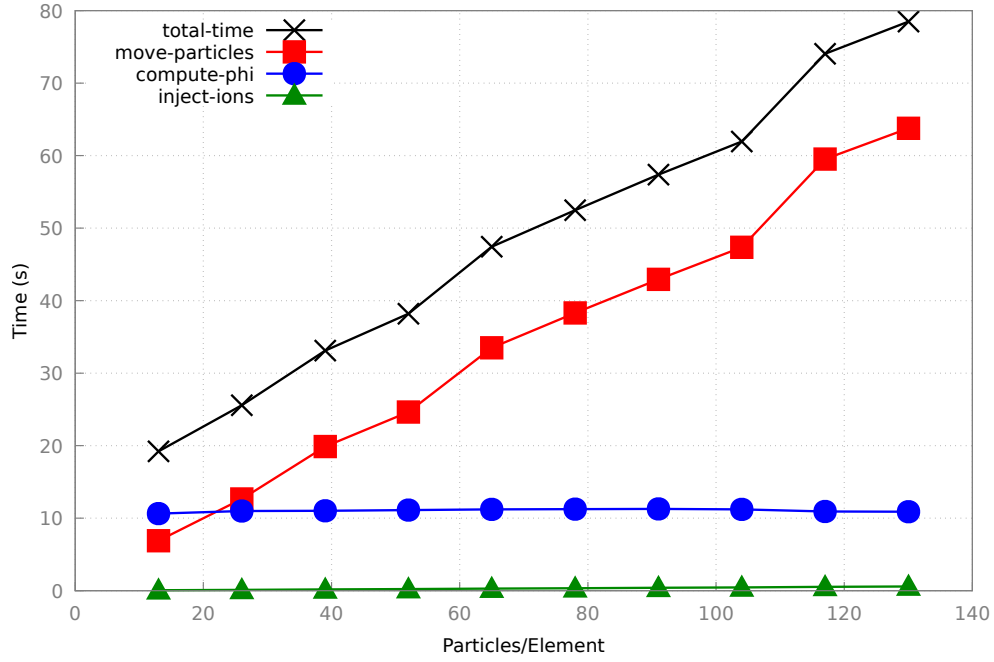
Figure 5: Mesh for the test system: Deuterium flow through a pipe.

The mini-application is run on a test system, consisting of Deuterium ion flow through a pipe, as shown in Figure 5. The pipe is 4 mm in length with a 1 mm radius, and is divided up into 9337 elements with an average edge length of ~ 0.2 mm. Faces on one end of the pipe are designated as inlet faces and the outer wall is fixed at a higher potential to retain the ions within the pipe.

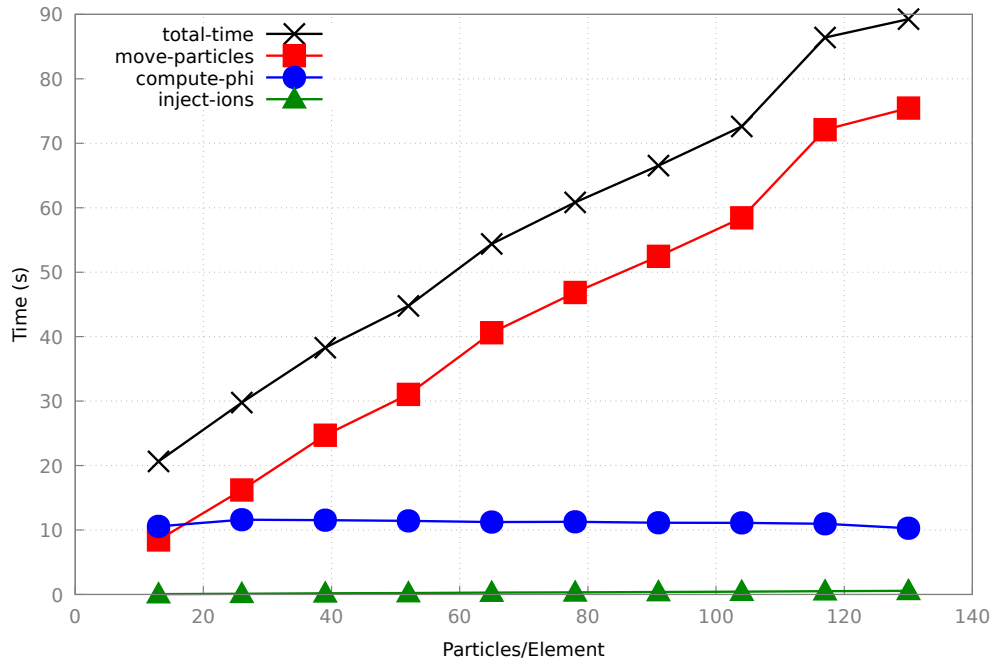
The plasma is fixed at $2 \times 10^8 K$ and the ions are injected with an input velocity of $1 \times 10^8 m/s$. In order to evaluate how the code scales as the number of ions change, 10 calculations are run for a range of ion densities between 2×10^{15} ions/ m^3 and 2×10^{16} ions/ m^3 . This equates to around 10-130 particles per element on average, or around 120,000 to 1,200,000 particles in total.

The system is run on a single core of the Viking cluster at York, which contains Intel(R) Xeon(R) Gold 6138 CPUs. The application is compiled with both the GNU C++ 11.3.0 and the Intel OneAPI C++ compiler version 2022.1.0 and the results are compared.

Figures 6(a) and 6(b) show the performance breakdown of the key routines in the mini-pic proxy app. In the case of this system, the majority of the time is spent in the `MoveParticles` routine for any more than around 20 ions per element. This time continues to grow approximately linearly with the number of particles in each element. As expected, the cost of `ComputePhi` does not vary significantly, as this only depends on the number of elements in the simulation. Compared to previous versions of the program, the cost of injecting new particles each iteration is now significantly smaller than the cost of moving the particles, also growing linearly with the number of particles being injected.



(a) GNU compilers



(b) Intel compilers

Figure 6: Timing breakdown of the key routines in the mini-pic application, for both the Intel and GNU compilers running on the Viking cluster.

References

- [1] Matthew T. Bettencourt, Dominic A. S. Brown, Keith L. Cartwright, Eric C. Cyr, Christian A. Glusa, Paul T. Lin, Stan G. Moore, Duncan A. O. McGregor, Roger P. Pawlowski, Edward G. Phillips, Nathan V. Roberts, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. EMPIRE-PIC: A Performance Portable Unstructured Particle-in-Cell Code. *Communications in Computational Physics*, x(x):1–37, March 2021.
- [2] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation*. Plasma Physics Series. Institute of Physics Publishing, Bristol BS1 6BE, UK, 1991.
- [3] John M. Dawson. Particle Simulation of Plasmas. *Reviews of Modern Physics*, 55:403–447, Apr 1983.
- [4] 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.
- [5] 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, sep 2015.
- [6] Robert Bird, Nigel Tan, Scott V Luedtke, Stephen Harrell, Michela Taufer, and Brian Albright. VPIC 2.0: Next Generation Particle-in-Cell Simulations. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2021.
- [7] Benjamin Daniel Dudson, Peter Alec Hill, David Dickinson, Joseph Parker, Adam Dempsey, Andrew Allen, Arka Bokshi, Brendan Shanahan, Brett Friedman, Chenhao Ma, David Schwörer, Dmitry Meyerson, Eric Grinaker, George Breyiannia, Hasan Muhammed, Haruki Seto, Hong Zhang, Ilon Joseph, Jarrod Leddy, Jed Brown, Jens Madsen, John Omotani, Joshua Sauppe, Kevin Savage, Licheng Wang, Luke Easy, Marta Estarellas, Matt Thomas, Maxim Umansky, Michael Løiten, Minwoo Kim, M Leconte, Nicholas Walkden, Olivier Izacard, Pengwei Xi, Peter Naylor, Fabio Riva, Sanat Tiwari, Sean Farley, Simon Myers, Tianyang Xia, Tongnyeoel Rhee, Xiang Liu, Xueqiao Xu, and Zhanhui Wang. BOUT++, 10 2020.
- [8] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.
- [9] Particle In Cell Consulting LLC. Finite Element Particle in Cell (FEM-PIC). <https://www.particleincell.com/2015/fem-pic/> (accessed October 10, 2022), 2015.
- [10] Robert F Bird, Patrick Gillies, Michael R Bareford, Andy Herdman, and Stephen Jarvis. Performance Optimisation of Inertial Confinement Fusion Codes using Mini-applications. *The International Journal of High Performance Computing Applications*, 32(4):570–581, 2018.

- [11] 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.
- [12] Toshiaki Tajima. *Computational Plasma Physics: With Applications to Fusion and Astrophysics*. CRC press, 2018.
- [13] Thomas JR Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation, 2012.
- [14] Stewart J. Clark, Matthew D. Segall, Chris J. Pickard, Phil J. Hasnip, Matt I. J. Probert, Keith Refson, and Mike C. Payne. First principles methods using CASTEP. *Zeitschrift für Kristallographie - Crystalline Materials*, 220(5-6):567–570, 2005.