Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 340 (2018) 67–84

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Performance of a Second Order Electrostatic Particle-in-Cell Algorithm on Modern Many-Core Architectures

Dominic A. S. Brown,[1] Steven A. Wright, Stephen A. Jarvis

Department of Computer Science
University of Warwick
Coventry, UK

Abstract

In this paper we present the outline of a novel electrostatic, second order Particle-in-Cell (PIC) algorithm, that makes use of 'ghost particles' located around true particle positions in order to represent a charge distribution. We implement our algorithm within EMPIRE-PIC, a PIC code developed at Sandia National Laboratories. We test the performance of our algorithm on a variety of many-core architectures including NVIDIA GPUs, conventional CPUs, and Intel's Knights Landing. Our preliminary results show the viability of second order methods for PIC applications on these architectures when compared to previous generations of many-core hardware. Specifically, we see an order of magnitude improvement in performance for second order methods between the Tesla K20 and Tesla P100 GPU devices, despite only a 4× improvement in the theoretical peak performance between the devices. Although these initial results show a large increase in runtime over first order methods, we hope to be able to show improved scaling behaviour and increased simulation accuracy in the future.

Keywords: Particle-in-Cell; PIC; Second Order Algorithms; Many-Core; P100; KNL; K20; GPU; Broadwell

1 Introduction

The behaviour of plasmas within various environments is of great interest to the scientific community, especially within the area of fusion energy research. For example, ventures such as the ITER project in France aim to develop the world's largest magnetic confinement fusion experiment. However, conducting these experiments can be both dangerous and prohibitively expensive. Therefore a common use of modern high performance systems is the simulation of plasmas under various conditions. The Particle-in-Cell (PIC) method is commonly used to carry out such simulations.

[1] Email:Dominic.Brown@warwick.ac.uk

https://doi.org/10.1016/j.entcs.2018.09.006

The emergence of modern many-core architectures that offer an extreme level of parallelism makes methods that were previously infeasible due to computational expense now viable. PIC codes often fail to fully leverage this increased performance potential due to their high use of memory bandwidth. The use of higher order PIC methods may offer a solution to this by improving simulation accuracy significantly for an increase in calculation intensity when compared to their first order counterparts. This greater expense is accompanied with a growth in the amount of memory throughput required during the simulation.

In this paper we will show the performance of a second order PIC algorithm. Our implementation uses second order finite elements and particles that are represented with a collection of surrounding ghost particles. These ghost particles each have associated weights and offsets around the true particle position and therefore represent a charge distribution. We also test our PIC implementation against a first order algorithm on various modern compute architectures.

Specifically, the contributions of this work are the following:

– We present the outline of a novel second order, electrostatic, Particle-in-Cell algorithm developed within Sandia National Laboratories' EMPIRE-PIC code. The method implements the standard PIC procedures at second order, and makes use of 'ghost particles' in order to represent a smoother particle charge distribution;
– We compare the performance of our second order algorithm to its first order counterpart on a variety of compute architectures. These include conventional Intel CPUs, NVIDIA Tesla GPUs (Kepler and Pascal architectures), and Intel's Xeon Phi Knights Landing. In this analysis we consider metrics such as overall execution time, the execution time of various key PIC kernels, and the average memory bandwidth achieved by the application;
– We highlight key performance issues within EMPIRE-PIC in order to facilitate future research and performance tuning.

The remainder of this paper is structured as follows: Section 2 outlines the relevant previous work related to PIC codes, and higher order PIC methods; Section 3 introduces EMPIRE-PIC, and the fundamentals of the PIC algorithm; Section 4 contains a description of our new algorithm; Sections 5 and 6 define the test cases used to examine the performance of our algorithm, and present the findings obtained from these results; finally, in Section 7 we conclude the paper, and highlight potential avenues for future research.

2 Related Work

There are many production codes that make use of the PIC algorithm to model the behaviour of charged particles under the influence of fields. EPOCH is an electromagnetic PIC code that models a staggered Yee grid to represent the electric and magnetic fields [1,18]. This allows the code to use Yee's finite-difference time-domain (FDTD) method to solve Maxwell's equations. A leapfrog method is then applied in order to update the particle positions. In addition, EPOCH also makes

use of higher order interpolation schemes. Attempts at the optimisation of EPOCH through the use of mini-applications have also been made, with particle sorting and improving the amount of autovectorisation carried out by the compiler leading to a notable speedup [2].

The WARP code, developed at Lawrence Berkeley National Laboratory (LBNL) and Lawrence Livermore National Laboratory (LLNL) implements an explicit and three-dimensional PIC algorithm in order to model the behaviour of high intensity ion beams [6]. Like EPOCH, WARP applies a second order leapfrog method to advance particles, but only uses linear interpolation functions between the particles and the grid points. Instead of the FDTD method, an electrostatic Poisson solver is used to conduct the field solve. There have been several versions of WARP, these include a code capable of Adaptive Mesh Refinement (AMR), and both two-dimensionial and three-dimensional electromagnetic implementations.

Wang et al. present GTC-P, maintained by Princeton University, an attempt at the development of a modern, highly parallelised PIC code [16]. GTC-P solves the 5-dimensional Vlasov equations in toroidal geometry in order to conduct simulations of magnetic confinement fusion experiments within tokamak devices. The application employs multiple layers of parallelism and has been ported to a variety of modern compute architectures such as NVIDIA's CUDA, and Intel's Xeon Phi. The performance of GTC-P has been tested on many notable HPC systems such as Sequoia, Piz Daint, Titan, and Tianhe-2 [14, 15, 17]. This work found that the application performed well in terms of both strong and weak problem scaling on a variety of input sizes, with the largest problem consisting of 80 million individual grid points.

Decyk proposes a streaming PIC algorithm optimised for graphics processing units, and demonstrates results using a two-dimensional, electrostatic PIC code [4]. Similarly to EPOCH, in order to improve streaming access and data locality, particles are sorted based on location such that the grid points required by closely located particles are more likely to be kept in cache or registers. In the ideal case of a cold plasma where particles never leave their cells, the streaming algorithm showed a speedup of 30× per particle, per timestep, on GPUs when compared to the CPU version.

Kong et al. detail another GPU PIC implementation [11]. The significant contribution of this work is the proposal of a method to reduce the impact of write conflicts between threads when depositing charge from particles in the simulation back onto the grid nodes – one of the most significant performance bottlenecks within the PIC method. This is accomplished by splitting the problem space into equally sized clusters, and only allowing particles within non-adjacent clusters to deposit charge simultaneously, in what is essentially a 'colouring' approach. While this does not eliminate the need for atomics entirely, it does reduce the overall amount of atomic operations used.

Other authors have also conducted research into the application of higher order methods to the PIC algorithm. Pointon presents a second order, charge conserving algorithm for accumulating charge and current in electromagnetic PIC simula-

tions [13]. The algorithm uses second order weighting functions to deposit charge on all cells adjacent to the particle. At problem boundaries the charge weighting is smoothly changed from second to first order, becoming entirely first order once the particle is on the boundary itself. Through the use of a test problem that consists of a conducting sphere in the centre of a square grid it was shown that charge is exactly conserved within the bounds of round off error. However, the algorithm requires further testing to determine its performance on three dimensional problems with strong electric and magnetic fields.

Jacobs and Hesthaven show a PIC algorithm that is capable of handling 4th order problems that make use of unstructured grids [9]. In order to solve Maxwell's equations on an unstructured grid, a Galerkin method is used which partitions the problem space into non-overlapping, triangular finite elements [8]. The interpolation used is similar to a method proposed by the same author, but is extended to be of the correct order [10]. Particles are translated into a Eulerian frame in order to weight the effects of their charge back onto the problem grid. Similarly to the algorithm discussed above, this method is also charge conserving. Through testing the algorithm on a variety of both one and two dimensional cases the authors show that it is capable of simulating basic plasma phenomena while also offering a large amount of geometric flexibility.

Moon et al. also present a higher order, charge conserving algorithm for unstructured grids [12]. The method is unique in that charge conservation is achieved from first principles instead of using correction steps to remove error that has been accumulated in the charge values over time. The preservation of Gauss' law is shown both analytically and through numerical testing.

In this paper we present a novel second order, electrostatic, PIC method for unstructured grids. In contrast to previous work, our algorithm makes use of particles surrounded by a set of ghost particles in order to mimic a charge distribution. Our implementation makes heavy use of the Trilinos library [7] and uses Kokkos as its parallel programming model [5]. We also test the performance of our algorithm on a variety of modern many-core hardware including NVIDIA GPUs, Intel CPUs, and Intel's Knights Landing.

3 Background

EMPIRE-PIC is an unstructured PIC application developed at Sandia National Laboratories (SNL) that is capable of both electrostatic and electromagnetic plasma simulations in two and three dimensional spaces. It makes use of many of the packages of the Trilinos library [7], particularly its linear solvers, and uses Kokkos [5] as its parallel programming model. This makes the code usable on a wide variety of compute architectures without requiring alteration of the original program source code. These architectures include traditional CPUs and Intel's Knights Landing (KNL) using OpenMP for multithreading, and NVIDIA GPUs using CUDA.

3.1 The Particle-in-Cell Method

The PIC method implemented within EMPIRE-PIC is used to model the behaviour of plasmas through representation of an electric field ($\boldsymbol{E}$) and magnetic field ($\boldsymbol{B}$) on an unstructured grid. The values of these fields are stored at each of the grid nodes, with interpolation being used to determine the field values at a specific point within a cell. The simulation also contains discrete particles that represent the plasma being simulated. Each of these is associated with a position on the grid, an electrical charge, a mass, and a velocity.

The standard PIC algorithm essentially consists of four main computational steps. These are summarised by the flow chart in Figure 1. The first of these is solving for the new values of $\boldsymbol{E}$ and $\boldsymbol{B}$ at the current timestep. In order to do this we must solve Maxwell's equations. However, due to the electrostatic approximation we only need to solve Gauss' law:

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

Where ρ is the charge density, ϵ_0 is the permittivity of free space, and ∇ is the divergence operator. Using a Finite Element Method (FEM) we then generate a stiffness matrix S which can be solved through a variety of iterative or direct techniques in order to obtain the values for electric potential ϕ. The electric field can then be determined via the following equation, where $\hat{v}$ represents the basis function for the type of finite element being used:

$$\vec{\boldsymbol{E}} \approx \sum_{j=0}^{N} \phi_j \nabla \hat{v}_j \tag{2}$$

Once the values of the electric field at the grid nodes have been obtained, it is necessary to determine the exact value of $\boldsymbol{E}$ at the position of each particle via interpolation before updating their velocity and position. In structured codes this is often accomplished by using simple methods such as area weighting (or volume weighting in three dimensions). This is especially applicable to those codes that make use of staggered grids such as that proposed by Yee [18]. However, in the unstructured case we must apply a different approach as grid cells can take arbitrary shapes. Within EMPIRE-PIC we gather the electric field to the particle position as described in Equation 3, where $\hat{e}$ is the relevant basis function, and x_i is the particle position.

$$\vec{\boldsymbol{E}}(x_i) = \sum_{j=0}^{N} \boldsymbol{E}_j \hat{e}_j(x_i) \tag{3}$$

The next step is to update the velocity of the particles within the simulation, which can be found by determining the force applied to the particles by the fields present. This force is described by the Lorentz force law:

$$\vec{\boldsymbol{F}} = \frac{q}{m}(\vec{\boldsymbol{E}} + \vec{v} \times \vec{\boldsymbol{B}}) \tag{4}$$

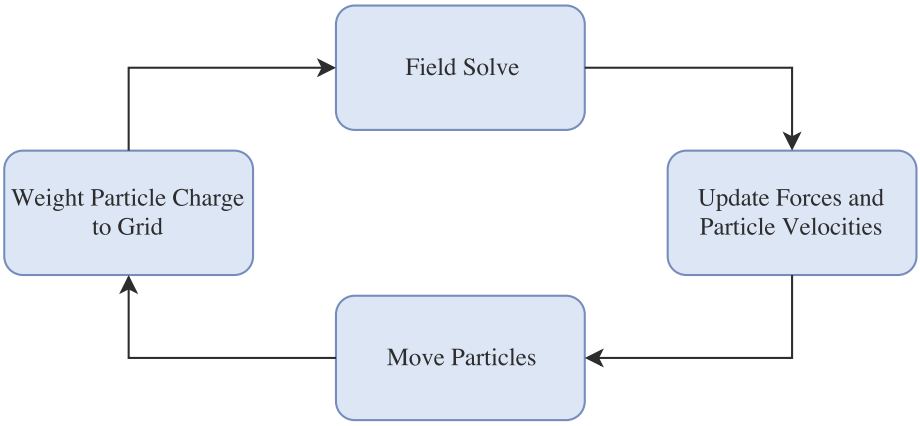


Fig. 1. Flow chart summarising the key components of the PIC algorithm

The standard method for solving for the Lorentz force and hence updating particle velocity within PIC codes is known as the Boris Method [3] (sometimes referred to as the Boris Pusher). In our electrostatic case, $\vec{\boldsymbol{B}} = 0$, so the calculation reduces to a multiplication of the charge to mass ratio of the given particle, and the value of the electric field at its current location. Once the velocity has been determined, we can update the particle position trivially by multiplying by the timestep size to calculate how much the particle should be moved. In the event of crossing an element boundary, the move is broken up into segments and the process is applied to each of these segments in turn.

The final step of the core electrostatic PIC algorithm is for the particles to deposit their contribution of electrical charge back onto the grid nodes before the beginning of the next field solve. The process is almost identical to that of weighting the fields to the particles: basis functions are used to calculate how much charge a particle must contribute to each node, and the relevant nodes are updated accordingly. However, the node updates must be carried out using atomic addition operations in order to prevent multiple threads attempting to update the charge value of the same node simultaneously, which would lead to erroneous results.

4 Algorithm

In this section we present the outline of our second order algorithm along with some of its implementation details. Pseudocode for this algorithm is shown in Algorithm 1. Before summarising the second order algorithm implemented within EMPIRE-PIC, it is first necessary to introduce the concept of both first and second order finite elements. Consider a two dimensional grid made up of quadrilateral elements. As outlined in Section 3 the values of the electric field are stored at the nodes of each element of this grid. In the first order situation nodes are situated at each of the four corners of the cell, and basis functions must be used to interpolate the value of $\boldsymbol{E}$ at a specific location within the cell.

At second order we must still conduct interpolation to determine exact grid values. Using a second order element gives us more points with which to interpolate

Algorithm 1 Pseudocode of our second order PIC algorithm

```
Conduct initial field solve for E
for i ← 0 to timesteps do
    for each central particle do
        Accumulate field values from each associated subparticle
    end for
    for each central particle do
        Update velocity with the Boris method
    end for
    for each central particle do
        Update position based on new velocity
    end for
    for each central particle do
        Deposit charge from each associated subparticle
    end for
    Conduct field solve for E
end for
```

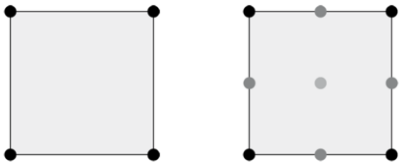
values from, thus leading to a more accurate solution. However, this increased accuracy comes at a significantly higher computational cost with a smaller increase in the required memory bandwidth. For clarity, Figure 2 shows an example of both a first and second order quadrilateral finite element, where the highlighted points represent the interpolation points for the electric field.

To raise the core algorithm of EMPIRE-PIC to this increased level of accuracy, the majority of the components of the traditional PIC algorithm were modified to make use of these second order elements. The one exception to this is that of the field solve, to which standard FEM can be applied as before, though the calculation is now more complex.

A unique feature of our algorithm is the use of 'ghost particles' in order to represent a smooth charge distribution. Unlike traditional PIC where each particle occupies a location on the grid, each physical particle is replaced by a set of ghost particles where the central particle occupies the 'true' position. Each of these is associated with a weight determining the proportion of the total charge carried by the ghost particle, and an offset which determines its position relative to the main central particle. Figure 3 shows a simple example of a particle in our algorithm.

4.1 Field Weighting to Particles

When weighting the values of $\boldsymbol{E}$ determined by the field solve to our new type of particle we follow a process very similar to that defined in Section 3, multiplying the electric field by the value of the element's basis function at the particle position. However, in our second order algorithm, we must accumulate the contribution of the electric field to all of the subparticles associated with the particle being processed before we can determine the correct value of the field at the position of the main central particle. To do so, we first obtain the basis function values at the position



(a) First order (b) Second order

Fig. 2. Example of first and second order quadrilateral finite elements

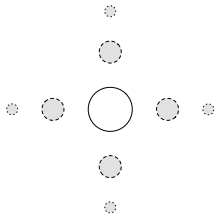


Fig. 3. An example of a particle in our second order algorithm. Ghost particles are shown in grey, and the true particle position in white

of the subparticle (through addition of its offset to the central particle position) which are then multiplied by the electric field value as normal. This result is then weighted according to the weight associated with the specific subparticle. In this way we compute the value of the electric field felt by each subparticle, it is then trivial to obtain the value on the true particle by simply summing the contributions of each subparticle for each grid dimension.

4.2 Particle Move

The use of the Boris push to compute the velocity of the particles is unchanged within our algorithm. This is due to the fact that all subparticles are positioned at a specific offset from the true particle position. Therefore we do not need to compute a new location for a subparticle based on its velocity – we can simply obtain it from its offset and the location of the central particle. However, we cannot simply bypass the subparticles during the particle move routine as it is necessary to update which finite element each subparticle is located in. This is accomplished by taking the velocity of the central particle and position of the subparticle, and then using these to determine whether the motion of the subparticle will lead to it staying in the same element, or crossing a boundary into an adjacent element. The element location of the subparticle is then updated accordingly. While this does lead to some redundant computation of data, the amount of calculations conducted is small thus leading to a very low impact on application performance. The position of the central particle is updated using the previously computed velocity, as normal.

4.3 Weighting Charge to the Grid

Much like the traditional PIC algorithm, particles in our second order method must deposit their charge contribution back to the grid nodes before the next field solve can be carried out. This follows a procedure much like our field weighting process, except in reverse. For each particle, each of its subparticles deposit their charge individually. As usual we obtain the subparticle position from its offset in order to determine the values of the element basis function at its location. We then multiply the particle charge by its weight, the basis function values, and the amount of physical particles that the particle is representing within our simulation before depositing it at each of the grid nodes. It should be noted that our algorithm still suffers from the issue of race conditions between threads when depositing this charge. Currently this issue is resolved through the use of atomic operations as in many other codes, though this leads to a performance overhead due to reduced parallelism.

5 Experiment Setup

In order to examine the peformance of our algorithm we compare its behaviour to that of the first order implementation. To this end we use a two dimensional two stream instability problem as our primary test case. This consists of two beams of electrons travelling in opposite directions separated by a stationary stream of positively charged ions. This leads to an increase in the energy of plasma waves. We also present performance results for Landau damping problems, where we simulate the effects of exponentially decreasing the energy of longitudinal waves in a charged plasma environment. This can be thought of as the inverse of the described two stream scenario.

Our largest test cases use a total of 1 million particles on a grid consisting of 64 cells in the x dimension, and 2 in the y dimension, for a total of 500 timesteps. This leads to a large amount of particles per grid cell, facilitating the later investigation of the effects of the overhead of atomic operations on the performance of the algorithm. Besides this, we use simulations with a variety of particle counts to assess the scalability of our PIC implementation. In addition to total execution time, and the timing of various key kernels, we also examine the average memory bandwidth used by the application at both first and second order in order to establish whether the move to second order drastically alters the degree to which the code is memory bound.

We test EMPIRE-PIC on the following compute architectures: NVIDIA Kepler GPUs, NVIDIA Pascal GPUs, Intel Broadwell CPUs, and Intel's Knights Landing (KNL). Information on the hardware used in this work can be found in Table 1. For the purposes of this work, all systems used consist of a single node. For the experiments run on the Broadwell system, we used 28 threads, with threads bound to cores, and hyperthreading disabled. On the KNL we found that using bound threads, with two hyperthreads per core, lead to the best results, therefore we used a total of 128 threads for the KNL experiments.

System	Peak TFLOPS	Mem. Bandwidth GB/s
Intel Xeon E5-2660 v4 Dual Socket (Broadwell)	0.448	102
Intel Xeon Phi 7210 (KNL)	3.0	475
NVIDIA Tesla K20	1.17	208
NVIDIA Tesla P100	4.7	732

Table 1
Table showing performance data of systems used to test EMPIRE-PIC

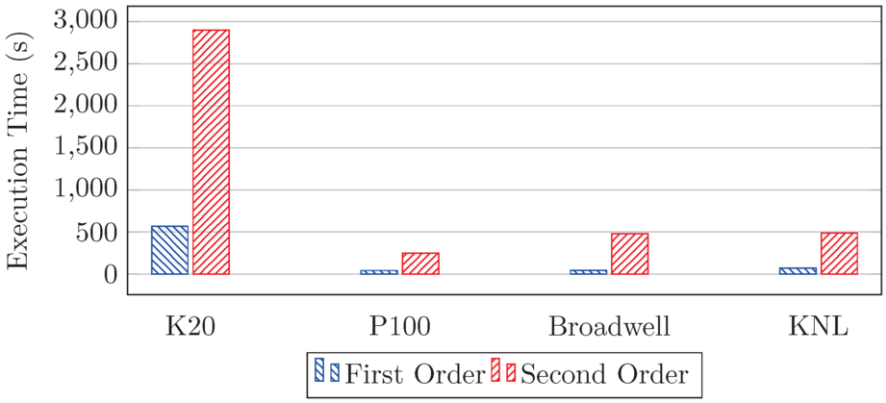


Fig. 4. Execution time of a two stream problem using 1 million particles, 500 steps

6 Results

6.1 Two Stream Problem

In order to obtain a general overview of the performance of EMPIRE-PIC on the two stream problems, we first compare the total execution time of the code on all of the architectures described in Table 1. Figure 4 shows the timings obtained from running the simulations at both first and second order levels of accuracy. It is clear to see that the more modern systems perform much better than the Tesla K20 when considering both first and second order problems. Specifically we see an order of magnitude difference between the K20 and P100 at second order, despite the theoretical peak performance of the P100 being only 4× greater than that of the K20. One possible explanation for such a difference is the hardware implementation of atomic operations on the Pascal architecture, as opposed to the lock-based software implemented atomics of Kepler. As a result of this, the heavy use of atomic operations when weighting particle charge back onto the grid nodes would be much more noticeable on the Tesla K20.

While the KNL performed approximately half as well as the P100 at second order, it performs almost as well at first order. Despite the disparity between the KNL and P100, the KNL performs on par with the Broadwell CPUs, taking only marginally longer on both first and second order experiments. Figure 5 shows the time taken to execute the key kernels of the code on both first and second order experiments, where each kernel represents one of the four phases of the PIC algorithm. On all of the architectures used, we see a noticeable increase in execution time in

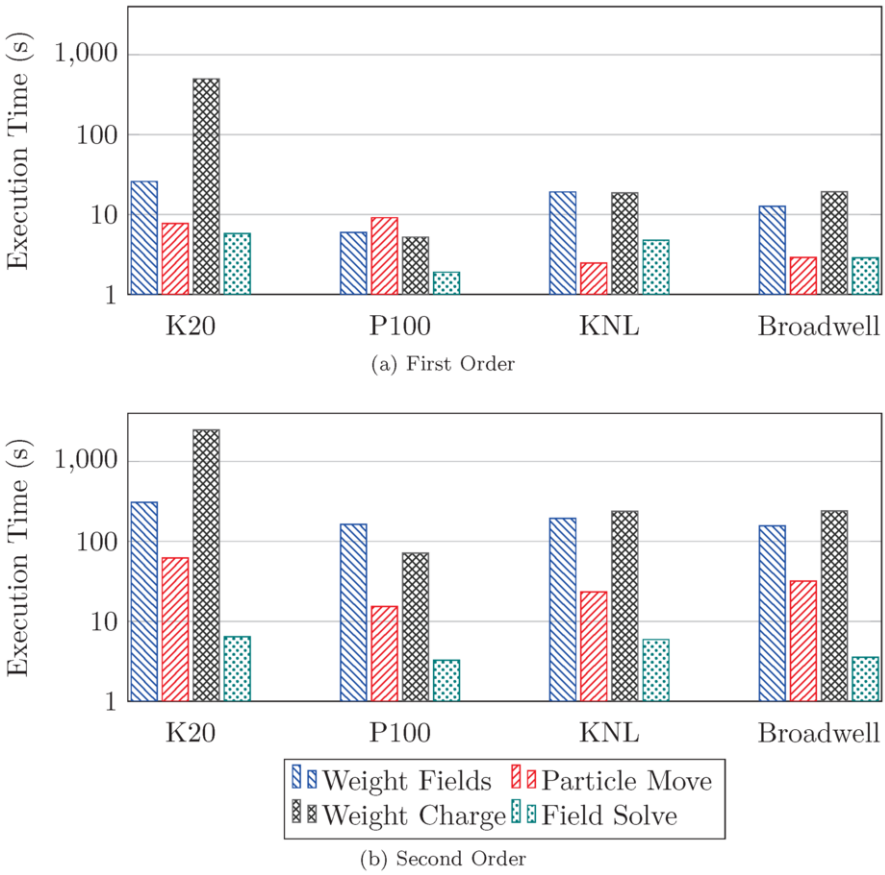


Fig. 5. Execution time of the key EMPIRE-PIC kernels for two stream experiments

the kernels responsible for weighting the fields to the particles, and for weighting the particle charge back onto the grid. As each of these kernels require basis functions to be evaluated at the position of each particle this difference is unsurprising as second order basis functions are more expensive to evaluate than their first order counterparts. The growth in the amount of charge deposits required due to the existence of ghost particles also contributes to this increase as the number of atomic operations carried out per timestep is larger. This overhead is less noticeable on the P100 when compared to the other three systems, again because of the hardware implementation of atomic operations. Also responsible for the high performance cost of the kernels that perform weighting operations is the expense of converting physical points to reference points in order to evaluate the basis functions. This is an iterative process involving matrix computation and is therefore expensive even for simple geometry. With regards to the field solve we observe almost no change in the amount of time taken to solve for $\boldsymbol{E}$, with disparities of approximately 1 second across all systems. This is to be expected as the method used is essentially the same, instead we are simply using a slightly more complicated element with a larger number of nodes.

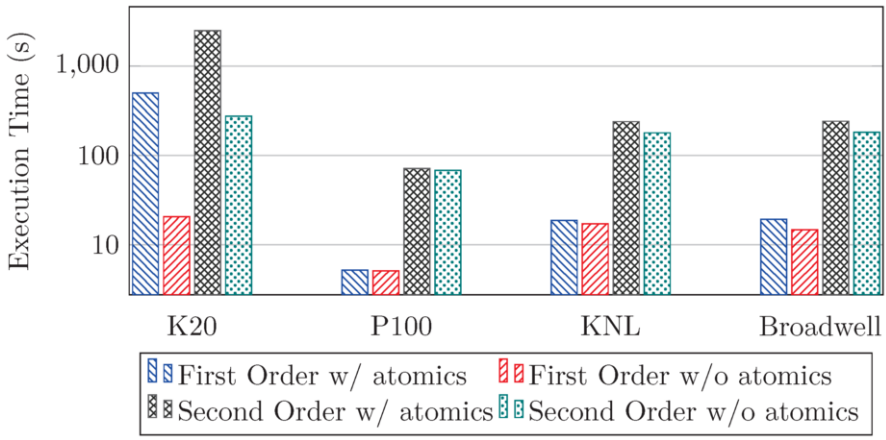


Fig. 6. Execution time of the weight charge kernel of EMPIRE-PIC, with and without atomics enabled

To assess the overhead of atomic operations on the charge weighting kernel, we also conducted experiments where the atomic access restriction to the charge array was disabled. We note that while this produces an incorrect simulation result due to the ensuing data hazards, it does provide insight into kernel and application performance. Figure 6 shows the time taken to execute the weight charge kernel with and without atomics enabled at first and second order. At first order we see the most pronounced difference on the Tesla K20 with a performance difference of greater than one order of magnitude. However, we see much smaller differences on the remaining systems with the difference in execution time on the Tesla P100 being indistinguishable from machine noise. At second order the differences remain visible, with the K20 continuing to show an order of magnitude difference in runtime. A performance gap also becomes far more noticeable on both Broadwell and the KNL. The disparity on the P100 remains small, highlighting the impact of hardware implemented atomic operations. These findings suggest that the issue of fast charge deposition in electrostatic PIC is less relevant on modern NVIDIA accelerators. However, a solution to this problem would be valuable to PIC application performance on all modern architectures – including the P100, as the impact is still nonzero.

We also consider how the execution time of EMPIRE-PIC varies as the amount of particles in the simulation is increased. Figure 7 shows the time taken for EMPIRE-PIC to run on various particle counts at first order and also using our second order algorithm. Unsurprisingly, the K20 scales much worse than the modern architectures, with its largest first order runs taking longer than the largest second order runs on all of the modern systems. At first order we see that the P100 performs slightly worse than the Broadwell system due to the overhead of transferring particle data from the host to the device. This difference decreases as the amount of particles grows because of the increase in the amount of compute required reducing the impact of the cost of data transfers on overall performance.

At second order the P100 scales much better than both Broadwell and the KNL, although all three systems are comparable. The improved P100 performance versus

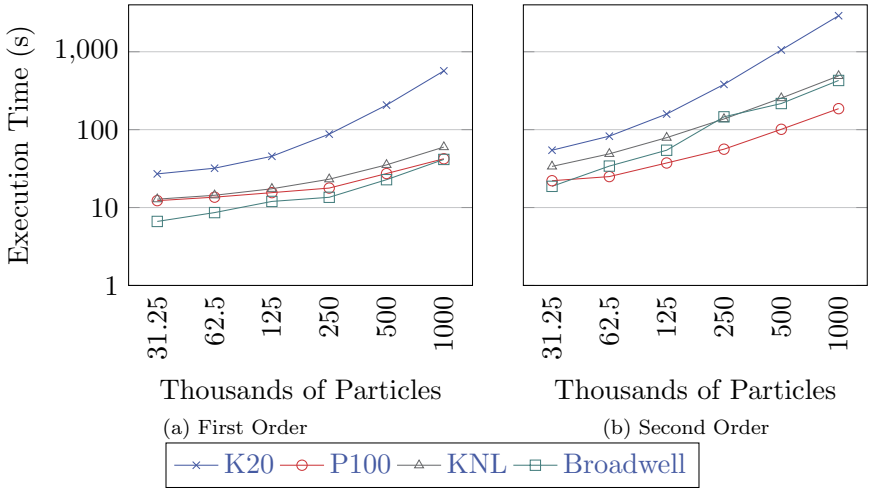


Fig. 7. Scaling results for the two stream experiments

System/Problem	Memory Bandwidth GB/s
K20 First Order	17.50
K20 Second Order	17.32
P100 First Order	29.12
P100 Second Order	45.88
KNL First Order	20.55
KNL Second Order	47.04
Broadwell First Order	5.93
Broadwell Second Order	4.73

Table 2
Table showing average memory bandwidth of EMPIRE-PIC on all systems at first and second order

Broadwell is brought about by the increase in FLOPs per byte moved from DRAM due to the higher floating point intensity of the second order method. As can be seen from Figure 7b, the P100 overtakes the Broadwell system at relatively low particle counts, in contrast to the similar behaviour at first order.

The Intel systems show very comparable scaling, with the initial gap between the two systems narrowing as problem size is increased. Therefore, while the non-accelerator systems are competitive with each other, on our algorithm the P100 is clearly superior and it is reasonabe to conclude that this would be the case on much larger problem sizes. However, at first order all three modern architectures exhibit similar scaling with very little difference between the three. The scaling behaviour does suggest that if first order problem size were to continue to increase a performance gap would appear between the P100 and Intel systems.

While there is a notable disparity between the time taken to execute our second order algorithm when compared against first order runs at all problem sizes, it is clear to see that second order methods are far more viable on modern many-core hardware versus earlier generations of these systems.

Table 2 shows the average memory bandwidth achieved by EMPIRE-PIC in our

Fig. 8. Execution time of a Landau damping problem using 1 million particles, 500 steps

experiments. We see almost no difference in the bandwidth achieved by the Tesla K20, whereas there is a noticeable increase for second order problems on both the Tesla P100 and the KNL. However, these increases do not make the application significantly more memory bound due to the high bandwidth limit for both systems. Surprisingly we observe a decrease of approximately 1 GB/s in the achieved bandwidth on the Broadwell system at second order. This result warrants further investigation.

6.2 Landau Damping Problem

Figures 8-10 show the results of carrying out the same testing procedure used on the two stream problem on the Landau damping experiments. The Landau damping problems exhibit almost identical performance to those of the two stream problems with the exception of the particle move kernel. On all architectures the execution time of this kernel is increased by a noticeable amount. This can be explained by the fact that the physical space represented within the problem is smaller in size than that of the two stream inputs. Therefore because the number of cells remains the same, the amount of physical space represented by one cell is reduced. This leads to particles potentially crossing more boundaries within a single step, leading to the move routine being executed on average more times per particle (as a move is split into a series of smaller moves when a boundary is crossed).

Scaling behaviour is also consistent across both types of problem, with the Landau damping experiments scaling only slightly worse across all architectures at both first and second order accuracy. This decline in scaling ability is brought about by the increased execution time of the particle move. The effects of the growth are most visible in second order runs with high particle counts where the particle move is executed more often. Table 3 shows the average memory bandwidth achieved by EMPIRE-PIC for our Landau damping experiments. We see almost identical behaviour to the two stream problems for both the Tesla K20, Broadwell, and KNL, whereas the Tesla P100 shows similar results for both first and second order runs.

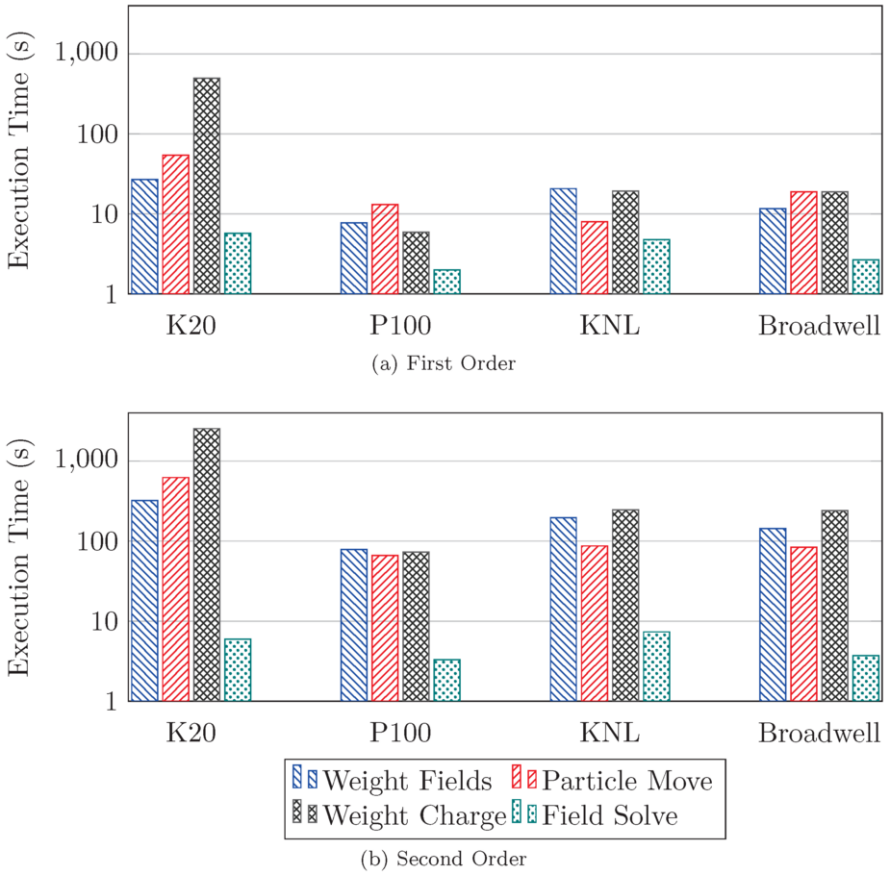


Fig. 9. Execution time of the key EMPIRE-PIC kernels for Landau damping experiments

System/Problem	Memory Bandwidth GB/s
K20 First Order	16.29
K20 Second Order	17.11
P100 First Order	43.31
P100 Second Order	44.89
KNL First Order	22.64
KNL Second Order	42.16
Broadwell First Order	8.19
Broadwell Second Order	6.56

Table 3
Table showing average memory bandwidth of EMPIRE-PIC on all systems at first and second order for Landau damping problems

7 Conclusion

In this paper we have presented a novel electrostatic, second order Particle-in-Cell algorithm and its implementation within EMPIRE-PIC, a PIC application developed at Sandia National Laboratories. We have examined the performance and scaling of this algorithm when compared to its first order counterpart on a vari-

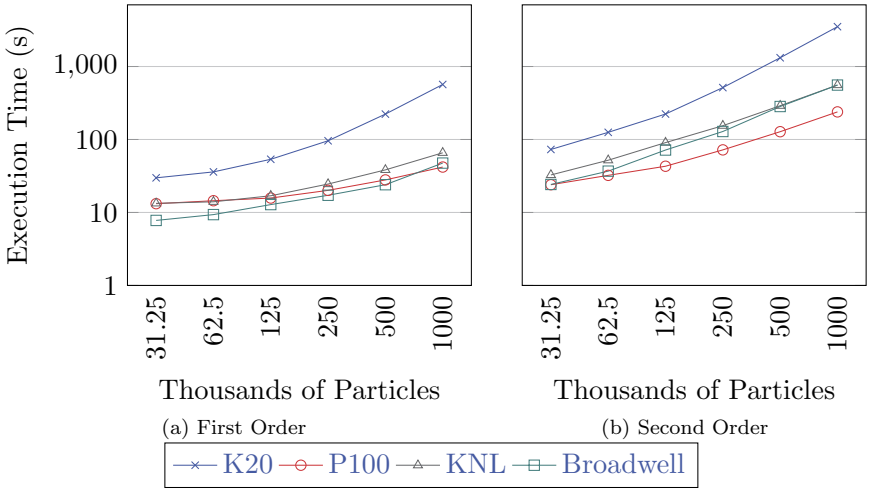


Fig. 10. Scaling results for the Landau damping experiments

ety of many-core systems including traditional CPUs, NVIDIA GPUs, and Intel's Knights Landing. Our results show an order of magnitude difference in performance between the Kepler and Pascal GPU architectures for second order problems, and also that both first and second order methods performed best on NVIDIA's Tesla P100. However, the other modern many-core systems remain competitive with each other and comparable with the P100. For example, Intel's Broadwell and KNL show almost identical performance at large particle counts for both two stream and Landau damping experiments. We have also shown that the move from first order to second order has not made the application significantly more memory bound due to the high bandwidth available on modern systems.

While second order methods currently perform and scale worse versus first order, our results suggest that second order PIC methods are becoming more viable on modern architectures when compared to previous generations of many-core hardware. We have also highlighted that, for our unstructured code, the significant performance issues are located within routines that involve weighting the values of the electric field to particle locations, or vice versa. This is both due to the usage of atomic operations, and also the conversion of physical points to reference points in order to determine basis function values.

7.1 Future Work

This is the first paper to document our electrostatic second order algorithm and its performance behaviour on various compute architectures. In future work we aim to use convergence analysis to quantify the accuracy improvement gained through using our method. Additionally, through resolution of performance issues we hope to show improved performance and scaling behaviours versus first order problems. Finally, we will extend our algorithm to function on electromagnetic simulations, where particles are influenced by both magnetic and electric fields.

Acknowledgement

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.

The authors would also like to express gratitude to Dr. Matthew Bettencourt of Sandia National Laboratories for his guidance and work on the project, and also for providing access to the EMPIRE-PIC source code.

References

[1] Arber, T. D., 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** (2015), p. 113001.

[2] Bird, R. F., P. Gillies, M. R. Bareford, J. A. Herdman and S. A. Jarvis, *Performance Optimisation of Inertial Confinement Fusion Codes using Mini-applications*, The International Journal of High Performance Computing Applications (2016).

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

[4] Decyk, V. K. and T. V. Singh, *Adaptable Particle-in-Cell Algorithms for Graphical Processing Units*, Computer Physics Communications **182** (2011), pp. 641–648.

[5] Edwards, H. C., C. R. Trott and D. Sunderland, *Kokkos: Enabling Manycore Performance Portability Through Polymorphic Memory Access Patterns*, Journal of Parallel and Distributed Computing **74** (2014), pp. 3202–3216, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[6] Grote, D. P., A. Friedman, J. Vay and I. Haber, *The WARP Code: Modeling High Intensity Ion Beams*, AIP Conference Proceedings **749** (2005), pp. 55–58.

[7] Heroux, M. A., R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams and K. S. Stanley, *An Overview of the Trilinos Project*, ACM Transactions on Mathematical Software **31** (2005), pp. 397–423.

[8] Hesthaven, J. and T. Warburton, *Nodal High-Order Methods on Unstructured Grids: I. Time-Domain Solution of Maxwell's Equations*, Journal of Computational Physics **181** (2002), pp. 186–221.

[9] Jacobs, G. and J. Hesthaven, *High-order Nodal Discontinuous Galerkin Particle-in-Cell Method on Unstructured Grids*, Journal of Computational Physics **214** (2006), pp. 96–121.

[10] Jacobs, G., D. Kopriva and F. Mashayek, *A Particle-tracking Algorithm for the Multidomain Staggered-grid Spectral Method*, in: *39th Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, 2001 .

[11] Kong, X., M. C. Huang, C. Ren and V. K. Decyk, *Particle-in-Cell Simulations with Charge-conserving Current Deposition on Graphic Processing Units*, Journal of Computational Physics **230** (2011), pp. 1676–1685.

[12] Moon, H., F. L. Teixeira and Y. A. Omelchenko, *Exact Charge-conserving Scattergather Algorithm for Particle-in-Cell Simulations on Unstructured Grids: A Geometric Perspective*, Computer Physics Communications **194** (2015), pp. 43–53.

[13] Pointon, T., *Second-order, Exact Charge Conservation for Electromagnetic Particle-in-Cell Simulation in Complex Geometry*, Computer Physics Communications **179** (2008), pp. 535–544.

[14] Tang, W., 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 Turbulence Simulations on Top Supercomputers Worldwide*, in: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16 (2016), pp. 43:1–43:12.

[15] Wang, B., 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: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13 (2013), pp. 82:1–82:12.

[16] Wang, B., S. Ethier, W. M. Tang, K. Z. Ibrahim, K. Madduri, S. Williams and L. Oliker, *Modern Gyrokinetic Particle-In-Cell Simulation of Fusion Plasmas on Top Supercomputers*, The International Journal of High Performance Computing Applications (2017).

[17] Wang, E., 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: *7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2016, pp. 25–32.

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