




Article

Energy Efficiency of a New Parallel PIC Code for Numerical Simulation of Plasma Dynamics in Open Trap

Igor Chernykh , Igor Kulikov , Vitaly Vshivkov, Ekaterina Genrikh, Dmitry Weins, Galina Dudnikova, Ivan Chernoshtanov and Marina Boronina 

Institute of Computational Mathematics and Mathematical Geophysics SB RAS, 630090 Novosibirsk, Russia

* Correspondence: chernykh@ssd.ssc.ru

Abstract: The generation of energy-efficient parallel scientific codes became very important in the time of carbon footprint reduction. In this paper, we briefly present our latest particle-in-cell code with the results of a numerical simulation of plasma dynamics in an open trap. This code can be auto-vectorized by the Fortran compiler for Intel Xeon processors with AVX-512 instructions such as Intel Xeon Phi and the highest series of all generations of Intel Xeon Scalable processors. Efficient use of processor architecture is the main feature of an energy-efficient solution. We present a step-by-step methodology of energy consumption calculation using Intel hardware features and Intel VTune software. We also give an estimated value of carbon footprint with the impact of high-performance water cooled hardware. The Power Usage Effectiveness (PUE) in the case of high-performance water cooled hardware is equal to 1.03–1.05, and is up to 1.3 in the case of air-cooled systems. This means that power consumption of liquid cooled systems is lower than that air-cooled ones by up to 25%. All these factors play an important role in the carbon footprint reduction problem.



Citation: Chernykh, I.; Kulikov, I.; Vshivkov, V.; Genrikh, E.; Weins, D.; Dudnikova, G.; Chernoshtanov, I.; Boronina, M. Energy Efficiency of a New Parallel PIC Code for Numerical Simulation of Plasma Dynamics in Open Trap. *Mathematics* **2022**, *10*, 3684. <https://doi.org/10.3390/math10193684>

Academic Editors: Leonid B. Sokolinsky and Mikhail Zymbler

Received: 28 August 2022

Accepted: 3 October 2022

Published: 8 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: high performance computing; particle-in-cell method; energy efficient algorithms

MSC: 68W10

1. Introduction

Last year, carbon dioxide emissions got more attention from many researchers in different areas. We can see a lot of scientific and popular papers in absolutely different journals, books, and on the internet. Of course, high-performance computing is associated with the carbon footprint theme because the most powerful supercomputers consume more than 15MW each, although the power consumption of each supercomputer is less than 1% of the typical aluminum plant consumption per year [1]. For example, we can read about the ecological impact of high-performance computing in astrophysics [2] or about the energy efficiency of openFOAM calculations in hydrodynamics [3]. Last decade we can see a lot of papers in computer science with different code optimization techniques [4,5]. Many years, our team have been developing parallel codes for plasma physics and for astrophysical hydrodynamics simulations. In [6] we presented energy efficiency calculation for our astrophysical code. We will use the same method as in [7] for energy efficiency calculation for a plasma physics code. This method considers the power consumption of compute nodes and the supercomputer's cooling system. First of all, we would like to say some words about the importance of plasma physics simulation and the Particle-in-Cell (PIC) method as one of the most common methods in this area. PIC simulation of high-beta plasmas in an axisymmetric mirror machine is of interest because of a new proposal of plasma confinement regime with extremely high pressure, equal to the pressure of the magnetic field (so-called diamagnetic confinement) [8]. This method allows essential decrease in the longitudinal losses. There are several theoretical papers devoted to different aspects of diamagnetic confinement: equilibrium and transport in the MHD approximation [8–10], influence of

kinetic effects on equilibrium and transport [11], and axisymmetric z-independence kinetic model of equilibrium [12]. Experimental demonstration of diamagnetic confinement in a mirror machine with high-power off-axis neutral beam injection (NBI) is planned in the Compact Axisymmetric Toroid (CAT) [13] device and Gas Dynamic Trap (GDT) at Budker Institute of Nuclear Physics [14]. These experiments are close to the experiments with high-power NBI in a field-reversed configuration, carried out on the C2-W/Norman device of the company Tri Alpha Energy. Using the numerical PiC code NYM for interpretation of the experiment in the C2-W device is worth noting [15]. The results of these experiments and calculations can be used for development of aneutronic fusion. In our work, we are using our own PIC algorithms and their parallel implementation. In this paper, we briefly describe the mathematical model, parallel implementation, and some numerical tests of the PIC code in Sections 1–3. A detailed description of the algorithms can be found in [16,17]. Sections 4 and 5 are dedicated to the energy efficiency calculation method and results.

2. Mathematical Model

The problem is considered on the basis of the of cylindrical trap sizes $[R_{max} \times L_{max}]$ with the magnetic field H . The fully ionized hydrogen plasma with zero temperature and density n_0 at the initial time $t = 0$ is inside the trap. At the time $t = 0$, injection of the particles into the trap from its center begins. We consider the injection as appearance of the ions and electrons at the injection point. The self-consistent non-linear interaction of the injected beam with the plasma and the magnetic field of the trap is the subject of research.

We assume axial symmetry of the problem, plasma quasi-neutrality ($n_i = n_e = n$), the negligibility of the displacement currents and the massless electron component and obtain the following equations in cylindrical coordinates with the following dimension units and scaling factors:

- time: $t_0 = 1/\omega$, where $\omega = \omega_{ci} = eH_0/cm_i$ is the ion gyrofrequency;
- length: $L_0 = c/\omega_{pi}$, where $\omega_{pi} = \sqrt{4\pi n_0 e^2/m_i}$ is the ion plasma frequency;
- velocity: the Alfvén velocity $V_A = H_0/\sqrt{4\pi m_i n_0}$.

The motion equations:

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i, \quad (1)$$

$$\frac{d\vec{v}_i}{dt} = \vec{F}_i, \quad (2)$$

where \vec{v}_i are the ion velocities, and \vec{r}_i are the ion coordinates.

$\vec{F}_i = \vec{E} + [\vec{v}_i, \vec{H}] - \kappa \vec{j}/n$ is the combination of the electromagnetic force with the force of the friction between the ions and the electrons.

The velocities of the electrons \vec{V}_e can be obtained from the equation:

$$\vec{j} = (\vec{V}_i - \vec{V}_e)n. \quad (3)$$

The distribution function of the ions $f_i(t, \vec{r}, \vec{v})$ defines their density n_i and mean velocity \vec{V}_i :

$$n(\vec{r}) = \int f_i(t, \vec{r}, \vec{v}) d\vec{v}, \quad (4)$$

$$\vec{V}_i(\vec{r}) = \frac{1}{n(\vec{r})} \int f_i(t, \vec{r}, \vec{v}) \vec{v} d\vec{v}. \quad (5)$$

The electric field \vec{E} is defined by the following equation:

$$\vec{E} + [\vec{V}_e, \vec{H}] + \frac{\nabla p_e}{2n} - \kappa \frac{\vec{j}}{n} = 0. \quad (6)$$

The electric field \vec{E} and magnetic field \vec{H} are defined from Maxwell's equations:

$$\frac{\partial \vec{H}}{\partial t} = -\text{rot} \vec{E}, \quad (7)$$

$$\text{rot} \vec{H} = \vec{j}. \quad (8)$$

The temperature T_e is defined by the following equation:

$$n \left(\frac{\partial T_e}{\partial t} + (\vec{V}_e \nabla) T_e \right) = (\gamma - 1) \left(2\kappa \frac{\vec{j}^2}{n} - p_e \text{div} \vec{V}_e \right) \quad (9)$$

for $\gamma = 5/3$ and the electron pressure $p_e = nT_e$.

The Equations (1)–(9) are based on the hybrid description of the process, where the ions are defined by the Vlasov kinetic equation and the electron component is described as a liquid with MHD equations. We use the particle-in-cell method [18,19], in which the particle coordinates, velocities, and charges are assigned to the particle data on the Lagrangian grid. The other functions (for example, the electric and magnetic fields, currents, temperature, and mean ion densities) are defined on their own staggered Eulerian grids. For plasma confinement with injection in the central domain of the trap and particle outflow through the trap ends $z = 0, z = Z_{max}$, the equilibrium parameters of the system require a detailed study. Due to the uneven particle distribution in the trap, the number of particles in some cell may be few orders greater than that in another cell, and the distribution is non-stationary on the early stages. The need in a large total number of particles (10^9) and long times of evolution ($\sim 10^4 \omega$) forces us to parallelize the complicated time-consuming algorithm.

3. Parallel Code

The parallel computing algorithm is based on the decomposition of the space and particle grids. We divide the computational domain into equal subdomains along the z-axis and assign the grid data of a subdomain to its own group of processor cores. The data of the ions located in a subdomain is divided among the cores of the group. On each time step, each core computes its particle trajectories, current, and charge densities. The master core of each group additionally gathers the current and charge densities, computes the functions on the Eulerian grid in view of the exchanges with the neighbour group cores and boundary conditions, and then broadcasts the results to all cores of its group.

In a group, the background ions for $t = 0$ and the injected ions on each timestep are distributed uniformly among the cores. When particles leave a subdomain, their data are sent to a core of the adjacent group, the receiving core for each sending core defined via the module operation. The algorithm provides an equal number of particles per core in group. The equal number of particles in groups is obtained due to periodic load balancing, the number of required cores in the groups is computed on the basis of the average number of particles in each core. The balancing is laborious and thus occurs periodically, for example, every 10^5 time steps. The dynamic load balancing allows speeding up the calculations.

We use explicit numerical schemes, and the trajectories of model particles and particle charge contribution can be calculated independently by each core. This internal parallelism of the particle-in-cell method is promising. However, there is a problem of the transition from the Lagrangian grid to the Eulerian one and back. At the same time, the computation of the force acting on each particle, the ion current, and the charge density via bilinear interpolation are the most time consuming procedures and may take more than 95% of the computation time. These difficulties are overcome via optimization of the algorithms for the calculations of the velocities, coordinates of ions, charge densities, current densities, and mean ion velocities to enable automatic vectorization [20]. We achieve $\approx 30\%$ speedup with the auto-vectorized code in comparison with the scalar code. This application of a single instruction to multiple data points allowed us to increase the efficiency of the computations and decrease their time. A detailed description of the model realization can be found in [21]. The pseudocode of the proposed parallel algorithm is presented in Appendix A.

The calculations were carried out with the Intel Xeon Phi 7290 processors of the Siberian Supercomputer Center, ICM&MG SB RAS, Novosibirsk, Russian Federation.

4. Numerical Simulation

For the simulations we use the magnetic field value $H_0 = H(0, L_{max}/2) = 0.2$ kg with the mirror ratio $H(0,0)/H(0, L_{max}/2) = 2$, length $L_0 = 22.8$ cm, and background density $n_0 = 10^{12} \text{ cm}^{-3}$. In the dimensionless units, the trap sizes are $R_{max} = 4$ and $L_{max} = 12$; the injection point coordinates are $R_{IP} = 0.4$ and $L_{IP} = 6$; the average beam speed for $t = 0$ is $|V| = 0.1V_A$, $V_A = 4.3 \times 10^8 \text{ sm/s}$; the initial angular velocity $V_\phi = 0.7|V|$. We used the spatial grid with 300×100 nodes, time step $\tau = 4 \times 10^{-6}$, and $J_b = 1.2 \times 10^5$ model particles. Each time unit $J_{inj} = 10^5$ model particles were injected, adding to the trap the charge of $Q_{inj} = 10^{18}$ real ions.

Figure 1 demonstrates the injected particle coordinates (z, r) at the times $T = 20$ and $T = 120$. The color of the particles denotes the time of injection of the particles. Figure 2 represents the magnetic field lines at the times $T = 20$ and $T = 120$. The injected particles begin to rotate in the magnetic field, forming a domain with the expelled magnetic field around the point of injection. The magnetic field inside this cavity reaches only few percents from the initial values, and thus new injected particles move in the cavity with their original velocities and change them on the border of the cavity. Then some particles move along the field lines towards the mirrors, and some particles remain captured in the cavity. The formation of the magnetic field cavity with the thin boundary layer of high field gradient may lead to the plasma confinement on the later evolution stages.

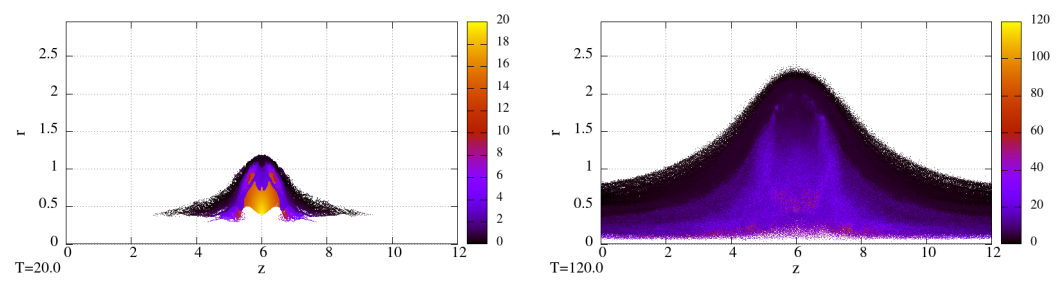


Figure 1. Injected beam coordinates at $T = 20$ (a) and $T = 120$ (b).

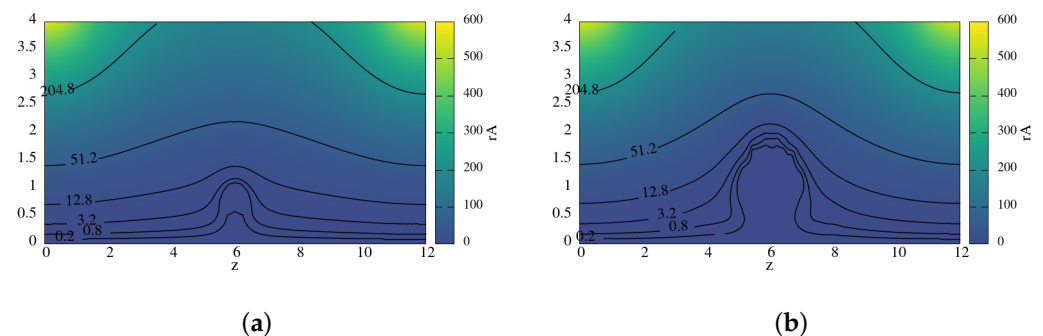


Figure 2. Magnetic field lines at $T = 20$ (a) and $T = 120$ (b).

5. Energy Efficiency

Recent initiatives on carbon footprint reduction will affect data centers. The carbon footprint is estimated from the energy consumption of the algorithm and the carbon intensity of the production of this energy: $\text{carbonfootprint} = \text{energyneeded} * \text{carbonintensity}$, where the energy needed is $\text{runtime} * (\text{powerdrawforcores} * \text{usage} + \text{powerdrawformemory}) * \text{PUE} * \text{PSF}$. The energy consumption of the computing cores depends on the model of CPU and the number of cores, while the memory power draw only depends on the size of

memory available. The usage factor corrects for the real core usage percentage. The default value of the usage factor is 1 for the full usage. The PUE (Power Usage Effectiveness) value measures how much energy is consumed by the cooling, lighting, etc. systems of the data center. The PSF (Pragmatic Scaling Factor) is used to take into account the code runs during debugging and final calculations. A detailed description can be found in [7]. Software engineers cannot control PUE and PSF values because they depend on the supercomputer architecture. However, we can create the energy-efficient code, which means that we have to use the CPU's architecture effectively. For modern CPUs from Intel or AMD, it means that we need to use vectorization instructions such as AVX2 or AVX-512 [22]. The best way is to optimize the code for autovectorization using a compiler, because this solution is more flexible towards the use of CPU intrinsics based on the specific processor architecture. We tested this approaches in our earlier works for astrophysical codes. Some of the results and software development techniques can be found in [23,24]. In the case of PIC code, in Listing A1 we provide a very simple pattern for aligning of main data. We used a C++ style aligning in a Fortran code, which can be compiled by the Intel Fortran compiler using the C++ precompiler. In our energy efficiency research, we used the Intel Xeon CPU architecture, because we can receive the energy consumption directly from hardware. This is a more precise approach than the use of approximate theoretical models from different publications. The Intel server platform also provides the possibility of DRAM power consumption collection for some DRAM models. The energy consumption of other devices inside the cluster's computational node is significantly less than that of processors and memory.

For the energy efficiency research, we used the Intel oneAPI software package [25]. It can measure the energy consumption data from processors, memory, and other hardware during calculations. We employed the Intel SoCwatch tool from Intel OneAPI for energy consumption evaluation. All collected data can be read and explained by the Intel VTune tool. We utilized the Intel Fortran Compiler, SoCwatch, and Intel Vtune from oneAPI 2022 for compiling and evaluating the energy consumption. In our research, we used RSC high-performance computing nodes of the NKS-1P cluster [26] with the Intel Xeon Phi 7290 (2nd KNL generation) and Intel Xeon 6248R (2nd generation of Intel Xeon Scalable) processors. Both type of CPUs support AVX512 instructions for vectorization. We also tested a workstation with the Intel core i9-10980XE processor, because this CPU also supports AVX-512 instructions. This building script from Listing A2 was used for our tests. Our code is an MPI code, and we utilize a standard mpirun for a Linux or mpiexec for Windows command for running the PIC simulation. As we said before, we use the `-cpp` key with some source files because we need to employ the C++ precompiler for aligning data and some other techniques to help the compiler to create auto-vectorized code. For the energy consumption collection we applied the SoCwatch tool, run by the command

$$socwatch - t60 - f power - rvtune - m - o results/test$$

as a separate process. SoCwatch produces an output file, which can be opened in the Intel VTune profiler, which shows the following data for analysis:

1. Analyze the amount of time spent in each sleep state (C-State) and processor frequency (P-State).
2. Identify which core spent time at what frequency.
3. Understand which cores were active during which timeframes during data collection.
4. Review the state residency by core, module, or package.
5. Explore how the state and frequency changed over time.

Figure 3 shows a C-state for the PIC code. We can see that in the first eight seconds of the numerical simulation, the cores are in state C2. It can also be considered as a transition state. The core clock is gated, and interrupts are not served. In this state, the PIC code is loading data for numerical simulation. After this, the CPU is going to a C0 state with some inactivity, which is connected with the internal data distribution between

some of the cores. The C0 state means that at least one hardware thread within the core is executing some task. In this state, cores stay active. Sometimes, in the case of thousand-core numerical simulation, this data can help to find bugs faster than other types of analysis, because we can find inactive cores. Finally, we can see that our PIC code uses all cores in an active state. For energy efficiency research, we need information about the performance in GFLOPS and the power consumption in Watts. For the performance evaluation, we used the Intel Advisor, which is also part of oneAPI. Detailed information, on how to evaluate the performance can be found in our previous work [27]. In this paper, we present data obtained from Advisor.

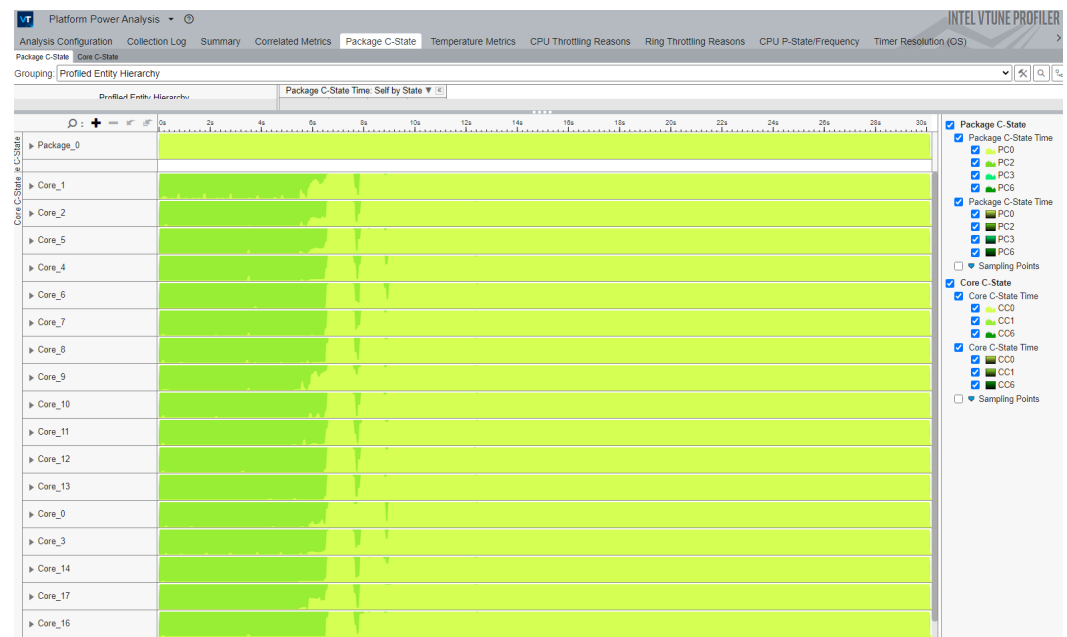


Figure 3. Intel Vtune Profiler C-state data for our PIC code.

Table 1 shows the energy efficiency of the developed PIC code on Intel Xeon Phi, the second generation of Intel Xeon Scalable processors, and Intel core i9 processor for workstations. We achieve $\approx 30\%$ performance growth with AVX512 instructions auto-vectorization in comparison with scalar maximum performance for each processor type. Why we concerning on aligning data in PIC code and trying to use other optimization techniques? Because the vectorization technique gives up to 8x performance speedup with 20–30% power consumption growth. This is the main feature of Intel Xeon scalable CPUs as well as other processors with AVX512 instructions support. This is hard to optimize memory bounded code to achieve maximum vectorized performance. We think that this is the main challenge for modern scientific parallel codes development. For the carbon footprint calculation, we need data from the cluster's cooling system. In our research, we used the NKS-1P system from the Siberian Supercomputer Center, the PUE factor of which is equal to 1.03. There are only 3% of energy consumed by the cooling system. This PUE is typical for the liquid-cooled systems (<https://rscgroup.ru/en/>, accessed on 24 August 2022). For this simple PIC test, the PSF is equal to 1. We used one node with two Intel Xeon 6248 CPUs for numerical tests, and one Intel Xeon Phi 7290 processor. It means that the *energyneeded* value = $48 \text{ h} \cdot 2 \cdot 193 \text{ W} \cdot 1.03 \cdot 1 = 19,084 \text{ Wh}$ for $2 \times$ Intel Xeon 6248R. In the case of one Intel Xeon Phi 7290 CPU, the *energyneeded* value = $48 \text{ h} \cdot 1 \cdot 202 \text{ W} \cdot 1.03 \cdot 1 = 9987 \text{ Wh}$. The total power consumption of the experimental setup is obtained by the node consumption * PUE factor. The carbon intensity from electricity generation can be found from [28], and for natural gas, it will be 0.91 pounds per kWh. In our case, we produced 17.29 pounds ($\approx 7.8 \text{ kg}$) of CO_2 from one test with double Intel Xeon 6248R and 9 pounds ($\approx 4 \text{ kg}$) with Intel Xeon Phi 7290.

Table 1. Energy efficiency of PIC code for numerical simulation of plasma dynamics problem in open trap. 1-CPU type, 2-Performance (GFLOPS), 3-Power Consumption (Watts), 4-Energy Efficiency (GFLOPS/Watts).

1	2	3	4
Intel Xeon 6248R (205 W, 24 cores, AVX512)	180	193	0.93
Intel Xeon 7290 (245 W, 72 cores, AVX512)	191	202	0.94
Intel core i9-10980XE (165 W, 18 cores, AVX512)	97.74	151	0.64

Unfortunately, we cannot compare these results with those of other teams, which are developing plasma physics codes yet, because the most common open-source software has different numerical schemes. We hope that the database of these authors [7] will help to do that in future.

6. Conclusions

In this paper, we briefly presented our latest PIC code, which is optimized for auto-vectorization by the Fortran compiler. We also present in this article a step-by-step instruction on how to calculate the energy consumption and carbon footprint of code. At the first step of code optimization, we achieved $\approx 30\%$ speedup of our auto-vectorized PIC code in comparison with parallel unvectorized code version. This speedup is achieved by the aligning data to help the Intel Fortran Compiler with auto-vectorization. The latest trends in reducing the carbon footprint dictate the need to pay attention to the quality of high-performance code development. The energy efficiency is measured in FLOPS/W. It is easy to understand that the most energy-efficient code has maximum FLOPS per watt. Unfortunately, the process of computational performance maximization is not such easy. Modern processors and accelerators are based on extended instructions for speeding up the calculations. Most of them are based on vectorization. If one does not use these instructions, the performance of the code will be unimpressed. For example, the scalar peak performance for Intel Xeon 6248R is about 171 GFLOPS. The dual precision vector peak performance is about 992 GFLOPS. The dual precision vector FMA peak is about 1984 GFLOPS for this processor. It means that vectorization can make a performance boost up to five times. In the case of vectorization and FMA instructions, the performance boost will be more than ten times. Another problem of performance maximization is the arithmetic intensity. The PIC method is a memory bounded problem. It means that this method has a very low FLOPS per byte of data value. This is why we have only 30% performance boost in our case. In future, we will optimize our mathematics for autogeneration of FMA instructions by the Fortran compiler. This approach can help us in additional performance boost. Unfortunately, now we cannot compare the energy efficiency of other PIC codes at this time, but we hope that there will be more energy efficiency research in future.

Author Contributions: Methodology, V.V., G.D. and I.C. (Igor Chernoshtanov); Software, M.B., I.C. (Igor Chernykh), I.K., D.W. and E.G.; Supervision, I.C. (Igor Chernykh); Validation, E.G., M.B. and G.D.; Visualization, M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Russian Science Foundation (project 19-71-20026).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The pseudocode of the proposed parallel algorithm:

Algorithm A1 PIC method parallel algorithm

```

Set initial conditions
for  $i = 1, 2, \dots, N_{\text{external}}$  do
  Rebalance
  for  $j = 1, 2, \dots, N_{\text{internal}}$  do
    if core contains injection point then Injection
    if core = master then Fields BCAST from masters to slaves
    New particle coordinates
    Boundary conditions for particles
    Particle exchanges with neighbours
    Particles sort among cells in one core
    Mean charge densities
    Mean current densities
    Reduce mean densities from slaves to masters
    if core = master then
      Mean densities exchange with neighbours
      Boundary conditions
      Currents
      Current exchange with neighbours
      Electric field
      Electric field exchange with neighbours
      Magnetic field
      Magnetic field exchange with neighbours
      Temperature
      Temperature exchange with neighbours

```

A typical pattern for aligning data (Listing A1) is as follows:

Listing A1. The pattern for aligning data in Fortran.

```

1. real*8, allocatable :: r(:)
2. real*8, allocatable :: z(:)
3. real*8, allocatable :: u(:)
4. real*8, allocatable :: v(:)
5. real*8, allocatable :: w(:)
6. real*8, allocatable :: q(:)
7. !dir$ attributes align:64 :: r
8. !dir$ attributes align:64 :: z
9. !dir$ attributes align:64 :: u
10. !dir$ attributes align:64 :: v
11. !dir$ attributes align:64 :: w
12. !dir$ attributes align:64 :: q

```

This listing shows how to help the Fortran compiler vectorize the code by aligning data.

Listing A2. Build script for our PIC code.

```

1. mpif90 -c limits.f
2. mpif90 -c types.f
3. mpif90 -c parallel.f
4. mpif90 -c algs.f
5. mpif90 -c -cpp push.f
6. mpif90 -axCORE-AVX512 -cpp hy.f parallel.o limits.o algs.o
   push.o

```


References

1. Energy Consumption in Aluminium Smelting and Changing Technologies Towards Gas Emission. Available online: <https://www.alcircle.com/specialreport/319/energy-consumption-in-aluminium-smelting-and-changing-technologies-towards-gas-emission> (accessed on 24 August 2022).
2. Zwart, S.P. The ecological impact of high-performance computing in astrophysics. *Nat. Astr.* **2020**, *4*, 819–822. [CrossRef]
3. Bonamy, C.; Lefèvre, L.; Moreau, G. High performance computing and energy efficiency: Focus on OpenFOAM. *arXiv* **2019**, arXiv:1912.03920.
4. Laros, J.H., III; Pedretti, K.; Kelly, S.M.; Shu, W.; Ferreira, K.; Dyke, J.V.; Vaughan, C. *Energy-Efficient High Performance Computing*, 1st ed.; Springer: London, UK, 2013.
5. Czarnul, P.; Proficz, J.; Krzywaniak, A. Energy-Aware High-Performance Computing: Survey of State-of-the-Art Tools, Techniques, and Environments. *Sci. Program.* **2009**, *2009*, 8348791. [CrossRef]
6. Kulikov, I.; Chernykh, I.; Karavaev, D.; Sapetina, A. The energy efficiency research of Godunov method on Intel Xeon scalable architecture. In Proceedings of the 2021 Ivannikov Ispras Open Conference (ISPRAS), Moscow, Russia, 2–3 December 2021.
7. Lannelongue, L.; Grealey, J.; Inouye, M. Green Algorithms: Quantifying the Carbon Footprint of Computation. *Sci. Prog.* **2019**, *8*, 8348791. [CrossRef] [PubMed]
8. Beklemishev, A.D. Diamagnetic “bubble” equilibria in linear traps. *Phys. Plasmas* **2016**, *23*, 082506. [CrossRef]
9. Beklemishev, A.D.; Khristo, M.S. High-Pressure Limit of Equilibrium in Axisymmetric Open Traps. *J. Plasma Fusion Res. Ser.* **2019**, *14*, 2403007.
10. Khristo, M.S.; Beklemishev, A.D. Two-dimensional MHD equilibria of diamagnetic bubble in gas-dynamic trap. *Plasma Phys. Control. Fusion* **2019**, *64*, 095019. [CrossRef]
11. Chernoshtanov, I.S. Collisionless particle dynamic in a diamagnetic trap. *Plasma Phys. Rep.* **2022**, *2*, 79–90. [CrossRef]
12. Kotelnikov, I.A. On the structure of the boundary layer in a Beklemishev diamagnetic bubble. *Plasma Phys. Control. Fusion* **2020**, *62*, 075002. [CrossRef]
13. Bagryansky, P.A.; Akhmetov T.D.; Chernoshtanov I.S.; Deichuli P.P.; Ivanov A.A.; Lizunov A.A.; Maximov A.A.; Mishagin V.V.; Murakhtin S.V.; Pinzhenin E.I.; et al. Status of the experiment on magnetic field reversal at BINP. *AIP Conf. Proc.* **2016**, *1771*, 030015.
14. Bagryansky, P.A.; Beklemishev, A.D.; Postupaev, V.V. Encouraging 382 results and new ideas for fusion in linear traps. *J. Fusion Energy* **2019**, *38*, 162–181. [CrossRef]
15. Belova, E.V.; Davidson, R.C.; Ji, H.; Yamada, M. Advances in the numerical modeling of field-reversed configurations. *Phys. Plasmas* **2006**, *13*, 056115. [CrossRef]
16. Chernykh, I.; Vshivkov, V.; Dudnikova, G.; Liseykina, T.; Genrikh, E.; Efimova, A.; Kulikov, I.; Chernoshtanov, I.; Boronina, M. High-Performance Simulation of High-Beta Plasmas Using PIC Method. *Commun. Comput. Inf. Sci.* **2020**, *1331*, 207–215.
17. Boronina, M.A.; Chernykh, I.G.; Dudnikova, G.I.; Genrikh, E.A.; Vshivkov, V.A. Mathematical modelling of beam dynamics in diamagnetic confinement regime of open trap. *J. Phys. Conf. Ser.* **2021**, *2028*, 012020. [CrossRef]
18. Hockney, R.W.; Eastwood, J.W. *Computer Simulation Using Particles*, 1st ed.; Hilger: Bristol, UK, 1988.
19. Boris, J.P. Relativistic Plasma simulation—Optimization of a hybrid code. In Proceedings of the Fourth Conference on Numerical Simulations of Plasmas, Washington, DC, USA, 2–3 November 1970; pp. 3–67.
20. Boronina, M.A.; Chernykh, I.G.; Genrikh, E.A.; Vshivkov, V.A. Performance improvement of particle-in-cell method for numerical modelling of open magnetic system. *J. Phys. Conf. Ser.* **2020**, *1640*, 012014. [CrossRef]
21. Boronina, M.A.; Chernykh, I.G.; Genrikh, E.A.; Vshivkov, V.A. Parallel Realization of the Hybrid Model Code for Numerical Simulation of Plasma Dynamics. *J. Phys. Conf. Ser.* **2019**, *1336*, 012017. [CrossRef]
22. Advanced Vector Extensions. Available online: https://en.wikipedia.org/wiki/Advanced_Vector_Extensions (accessed on 24 August 2022).
23. Kulikov, I.M.; Chernykh, I.G.; Glinskiy, B.M.; Protasov, V.A. An Efficient Optimization of HLL Method for the Second Generation of Intel Xeon Phi Processor. *Lobachevskii J. Math.* **2018**, *39*, 543–551. [CrossRef]
24. Kulikov, I.; Chernykh, I.; Karavaev, D.; Prigarin, V.; Sapetina, A.; Ulyanichev, I.; Zavyalov, O. A New Approach to the Supercomputer Simulation of Carbon Burning Sub-grid Physics in Ia Type Supernovae Explosion. *Commun. Comput. Inf. Sci.* **2022**, *1618*, 210–232.
25. Intel One API. Available online: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html> (accessed on 24 August 2022).
26. Siberian Supercomputer Center. Available online: <http://www.sssc.icmmg.nsc.ru/hardware.html> (accessed on 24 August 2022).
27. Chernykh, I.; Vorobyev, E.; Elbakyan, V.; Kulikov, I. The Impact of Compiler Level Optimization on the Performance of Iterative Poisson Solver for Numerical Modeling of Protostellar Disks. *Commun. Comput. Inf. Sci.* **2021**, *1510*, 415–426.
28. How Much Carbon Dioxide Is Produced per Kilowatthour of U.S. Electricity Generation? Available online: <https://www.eia.gov/tools/faqs/faq.php?id=74&t=11> (accessed on 24 August 2022).