



## Particle-in-Cell laser-plasma simulation on Xeon Phi coprocessors

I.A. Surmin<sup>a</sup>, S.I. Bastrakov<sup>a</sup>, E.S. Efimenko<sup>a,b</sup>, A.A. Gonoskov<sup>a,b,c</sup>, A.V. Korzhimanov<sup>a,b</sup>, I.B. Meyerov<sup>a,\*</sup><sup>a</sup> Lobachevsky State University of Nizhni Novgorod, Building 2, 23 Gagarina Avenue, Nizhni Novgorod, 603950, Russia<sup>b</sup> Institute of Applied Physics of the Russian Academy of Sciences, 46 Ul'yanov Street, Nizhni Novgorod, 603950, Russia<sup>c</sup> Chalmers University of Technology, SE-412 96, Gothenburg, Sweden

## ARTICLE INFO

## Article history:

Received 29 May 2015

Received in revised form

3 February 2016

Accepted 10 February 2016

Available online 17 February 2016

## Keywords:

Plasma simulation

Particle-in-Cell

Xeon Phi

## ABSTRACT

This paper concerns the development of a high-performance implementation of the Particle-in-Cell method for plasma simulation on Intel Xeon Phi coprocessors. We discuss the suitability of the method for Xeon Phi architecture and present our experience in the porting and optimization of the existing parallel Particle-in-Cell code PICADOR. Direct porting without code modification gives performance on Xeon Phi close to that of an 8-core CPU on a benchmark problem with 50 particles per cell. We demonstrate step-by-step optimization techniques, such as improving data locality, enhancing parallelization efficiency and vectorization leading to an overall  $4.2\times$  speedup on CPU and  $7.5\times$  on Xeon Phi compared to the baseline version. The optimized version achieves 16.9 ns per particle update on an Intel Xeon E5-2660 CPU and 9.3 ns per particle update on an Intel Xeon Phi 5110P. For a real problem of laser ion acceleration in targets with surface grating, where a large number of macroparticles per cell is required, the speedup of Xeon Phi compared to CPU is  $1.6\times$ .

© 2016 Elsevier B.V. All rights reserved.

## 1. Introduction

The progress made in high intensity laser pulse generation over the last 20 years has stimulated theoretical and experimental research on ultra-intense laser-matter interaction in extremely relativistic regimes [1,2]. This is valuable for both fundamental research in the physics of matter in extreme conditions and various applications. The noteworthy directions are the following: laser-driven electron acceleration to ultrarelativistic energies [3]; acceleration of ion beams to tens and hundreds MeV/nucleon [4,5]; generation of X-ray and gamma radiation, including pulses of atto- and zeptosecond duration [6]; quantum electrodynamics effects in ultrahigh intensity fields, namely electron-positron pair production and the nonlinear optics of the vacuum [7,8]. The applications are fast ignition of inertial confinement fusion targets [9], hadron therapy for cancer treatment [10], protonography, X-ray imaging [11], etc.

High intensity laser-matter interaction involves several non-linear physical phenomena: relativistic and ponderomotive self-focusing, collisionless heating, various plasma instabilities, high harmonics generation, and others. A proper analytical examination is only possible in several simple cases. Thus, one of the main tools for theoretical research is numerical simulation. Along with investigation of the underlying physics, computer simulation helps to set experiments by allowing faster and more efficient adjustment of parameters for laser and target, designing experimental schemes, and interpreting the results.

The most widely used method for simulation of plasma in ultrahigh fields is the Particle-in-Cell (PIC) method [12] that allows full 3D simulation including the main processes governing laser-plasma interaction. The computational complexity of the Particle-in-Cell method is relatively low compared to other kinetic methods (e.g. Euler [13]), yet large-scale 3D simulation requires high-performance implementation aimed at supercomputers. Currently, several widely known implementations of the fully relativistic Particle-in-Cell method are capable of 3D large-scale plasma simulation on supercomputers, most notably, OSIRIS [14], VPIC [15], VLPL [16], WARP [17], PIConGPU [18]. Continuous progress is made not only in terms of implementation efficiency, but also in algorithms. A striking example is the relativistic boosted frame method which allows the speeding up of simulations of laser propagation in low density plasma by a factor of tens [17].

\* Corresponding author.

E-mail addresses: [i.surmin@gmail.com](mailto:i.surmin@gmail.com) (I.A. Surmin), [bastrakov@vmk.unn.ru](mailto:bastrakov@vmk.unn.ru) (S.I. Bastrakov), [evgeny.efimenko@gmail.com](mailto:evgeny.efimenko@gmail.com) (E.S. Efimenko), [arkady.gonoskov@gmail.com](mailto:arkady.gonoskov@gmail.com) (A.A. Gonoskov), [korzhimanov.artem@gmail.com](mailto:korzhimanov.artem@gmail.com) (A.V. Korzhimanov), [meerov@vmk.unn.ru](mailto:meerov@vmk.unn.ru) (I.B. Meyerov).

Heterogeneous cluster systems are based not only on conventional CPUs, but also on different types of accelerators, including GPUs and the recently introduced Intel Xeon Phi coprocessors. This has led to a growing interest in Particle-in-Cell adaptations for such systems. While the Particle-in-Cell method is generally suitable for modern GPUs, its efficient implementation requires a meticulous approach to data structures and parallel processing schemes as well as conscious usage of different layers of GPU memory and is by no means a simple porting of an efficient implementation for CPUs.

The efficacy of Xeon Phi coprocessors in application to different domains of computational science has been demonstrated [19–21], and Particle-in-Cell codes are candidates for obtaining a good speedup on Xeon Phi compared to CPUs [22]. Straightforward porting can be done rather quickly, although it will be efficient only if the code has been properly optimized for CPUs and has a large degree of parallelism on the thread level and single instruction, multiple data (SIMD) level. This might be a limiting factor for codes that scale well up to 8–16 threads, but not up to 120–240 threads or are only capable of using low width SIMD. Thus, a no-effort “just rebuild” porting does not seem to be efficient except in some very special cases; at the same time porting with reasonable additional tuning seems promising.

This paper presents our experience in porting the existing parallel Particle-in-Cell plasma simulation code PICADOR [23,24] to Xeon Phi coprocessors. We demonstrate a step-by-step optimization process with iterative bottleneck analysis and application of optimization techniques. We believe that the encountered problems and applied optimizations are not specific to the Particle-in-Cell implementation and similar ideas are useful for a wide class of applications. The porting and optimization of the code are illustrated on CPUs and Xeon Phi coprocessors. We use a benchmark frozen plasma simulation problem with ideal balance and no MPI exchanges. The performance of the final version is then evaluated on a real problem of laser ion acceleration in targets with surface grating. This problem is shown to demand a large number of macroparticles which makes it suitable for Xeon Phi coprocessors. Our discussion is limited to a single-node case with 1 or 2 Xeon Phi coprocessors. While scaling to many nodes is very important, it is achieved by means of domain decomposition and efficient MPI communications that are almost independent of the optimization techniques for the single-node level considered in this work.

The paper is organized as follows. The Particle-in-Cell method is briefly described in Section 2. In Section 3 we discuss suitability of the method for Xeon Phi coprocessors and optimization methodology. We present experience of porting and optimization of the Particle-in-Cell plasma simulation code PICADOR to Xeon Phi in Section 4. Section 5 contains performance evaluation of a real physical problem.

## 2. Particle-in-Cell method

This section briefly describes the Particle-in-Cell method in the form used in our implementation, a detailed description is given in [12].

The simulation area is a 3D axis-aligned parallelepiped covered by a uniform spatial grid. The dynamics of the electric field  $\mathbf{E}$  and magnetic field  $\mathbf{B}$  is defined by Maxwell's equations solved on the staggered Yee grid [25] using the finite-difference time-domain (FDTD) method [26]. The plasma is represented as an ensemble of  $N$  charged macroparticles, each with variable momentum  $\mathbf{p}$  and position  $\mathbf{r}$ , and constant mass  $m$  and charge  $q$ . The position  $\mathbf{r}$  and velocity  $\mathbf{v}$  evolve according to Newton's law in relativistic form that is numerically integrated using the explicit Boris method [27] which is widely employed for pushing charged particles due to its good long-time accuracy [28]. The particle motion creates electric

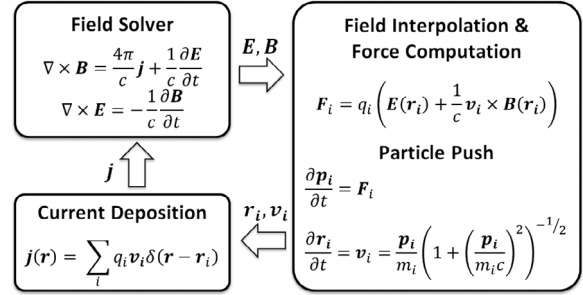


**Fig. 1.** Computational scheme of the Particle-in-Cell method. An iteration of the computational loop corresponds to a time step. The equations are given in the centimetre–gram–second system of units. Arrow labels denote data dependences between stages.

current  $\mathbf{j}$  that is a part of Maxwell's equations, completing the self-consistent system of equations.

A basic computational scheme of the Particle-in-Cell method with the main equations and data dependences is shown in Fig. 1. An iteration of the computational loop corresponds to a time step. Each time step consists of 4 main stages. The field solver updates the grid values of the electromagnetic field. Field interpolation from the grid to the particle positions is performed to compute the Lorentz force affecting the particles. For pushing the particles one step further, the equations of particle motion are solved. The computational loop closes with computation of the current created by the particle motion. In terms of software implementation, it is convenient to merge the field interpolation, force computation and solution of the equations of particle motion into one stage, referred to as the particle push. The Particle-in-Cell method can be extended in many ways [29]. In this paper we consider only the basic version of the method as these stages take a significant share of computational time even in more complicated simulations.

The Particle-in-Cell method operates with two principally different data sets: an ensemble of charged particles with continuous coordinates and values of the field and current density set on a discrete grid. The most time consuming stages of particle push and current deposition operate with both data sets, thus implementation of these stages plays a major role in both accuracy and computational efficiency.

## 3. Implementation of the Particle-in-Cell method for Xeon Phi coprocessors

In this section we analyze the suitability of the Particle-in-Cell method for Xeon Phi coprocessors and discuss the optimization techniques required to achieve good performance. There are several highly efficient implementations of the Particle-in-Cell method for GPUs [18,30] which implies that the method may be also suitable for Xeon Phi. However, there are several features and considerations which are specific to Xeon Phi.

### 3.1. Overview of Xeon Phi coprocessors

A Xeon Phi coprocessor has 60 cores (61 on some modifications) on shared memory with Linux on board. The cores are  $\times 86$ -compatible, but have a simplified architecture compared to CPU cores with no support to out-of-order execution and branch prediction, combined with wider 512-bit vector registers (256-bit on modern CPUs). Each core supports up to 4 hardware threads, it is recommended to run at least 2 threads per core [31]. The peak performance of a Xeon Phi core is lower than of a modern CPU core, but the larger number of cores provides overall nearly 1 TFLOPS ( $10^{12}$  floating-point operations per second) peak performance in

double precision. Another notable advantage over CPUs is 350 GB/s peak memory throughput, 5–10 times larger compared to CPUs.

Programming for Xeon Phi can be done using traditional programming languages and parallel programming libraries: C, C++, Fortran languages, MPI, OpenMP, Intel Cilk Plus, OpenCL, Intel Threading Building Blocks and Math Kernel Library. This significantly simplifies the porting of existing applications as the vast majority of the code (or even the whole code) does not need to be modified for Xeon Phi. There are three execution modes: running only on Xeon Phi as a multicore processor (native mode), running the main program on CPU and calling computational kernels on a coprocessor similar to GPU usage (offload mode), or running some MPI processes on CPUs and some on Xeon Phi coprocessors (symmetric mode) [31].

Overall, Xeon Phi has manycore architecture with wide vector registers and high throughput memory. Parallelism on both thread level (120–240 threads) and 512-bit SIMD level is crucial for good performance. Although existing applications can generally be ported without any significant effort, performance of such ports is not guaranteed even for implementations that are efficient on CPU. Some additional optimization and tuning is most likely required. However, the main optimization principles for CPU and Xeon Phi coprocessors are similar, so optimization for one of the platforms is likely to be beneficial for the other.

### 3.2. Analysis of suitability of the Particle-in-Cell method for Xeon Phi coprocessors

Generally there are several fundamental factors that determine the applicability of accelerators such as GPUs and Xeon Phi coprocessors. First, there is a memory limitation of 6–16 GB RAM per accelerator which is coupled to the host with a PCI Express bus. While this could be very restrictive for applications working with large sets of data, Particle-in-Cell simulations are spatially local, thus allowing efficient decomposition of a problem between many computational nodes and therefore fitting memory limitations of a node.

Performance-wise Xeon Phi coprocessors offer about 5× peak performance advantage over a multicore CPU. Achieving a significant share of peak performance is challenging on CPUs and even more so on Xeon Phi coprocessors. A half of peak performance of Xeon Phi is attributed to vector fused multiply–add (FMA) instruction that performs two floating-point operations at once. Obviously, real applications do not consist of sequences of pure FMA calls, this fact effectively reduces performance by half.

The efficient utilization of Xeon Phi requires excellent scaling on shared memory. The coprocessors offer 60 cores with up to 4 hardware threads. Well-suited applications usually either scale well up to 120–240 threads or use efficient hybrid MPI + OpenMP parallel scheme. One way or another, the method must have a significant parallelization potential. An ideal application in this regard (for pretty much all parallel hardware) is the Monte-Carlo method which allows ideal scaling. The basic version of the Particle-in-Cell method described in Section 2 also allows ideal scaling on shared memory CPU cores and, in our experience, 75% scaling efficiency on Xeon Phi cores (for a large number of particles per cell). Extensions of the Particle-in-Cell method, such as pulse generation, ionization and filtering, generally do not affect scaling as they allow parallel processing. A notable exception is accounting for the effects of quantum electrodynamics in strong fields [29]. Simulations involving electromagnetic cascades yield intensive electron–positron pair production in a very small area, hence very imbalanced particle distributions. A special approach to particle distribution between threads might be required in this case. It is important to note that good scalability is the necessary, but not sufficient, condition for efficient utilization of Xeon Phi.

Another important – and often the most challenging – factor is the utilization of vector units for performing floating-point operations, the so-called vectorization. Vectorization is important for CPUs, and even more so on Xeon Phi due to 512-bit vector registers compared to 256-bit on modern CPUs. Consequently, vectorization theoretically offers double dividend on Xeon Phi compared to CPUs and a lack of vectorization puts a double penalty on Xeon Phi performance. The Particle-in-Cell method has two stages that could be problematic for vectorization: field interpolation and current deposition, that are usually the most time-consuming parts of the method. A straightforward implementation of these operations in 3D results in a non-unit stride (not local) memory access pattern that limits vectorization efficiency. An additional obstacle for low-order particle form factors is relatively low computational density. Nevertheless, efficient vectorization of the Particle-in-Cell method can be performed [32].

Our analysis shows that the Particle-in-Cell method, although not ideally suited for Xeon Phi coprocessors due to vectorization issues, is a promising candidate. We will show that with proper programming a Xeon Phi coprocessor can achieve about 2× speedup over a tuned implementation on a modern 8-core CPU on a benchmark and in the simulation of a real physical problem.

### 3.3. Optimization methodology

Although some computational applications are developed for a specific class of hardware and thus could largely benefit from hardware-specific features, the vast majority of applications are initially developed for CPUs and then ported to accelerators. This subsection is devoted to a general discussion of porting and optimization of an existing computational application from CPUs to accelerators. We suppose the employed numerical schemes to be suitable for parallel processing and that the code is developed using standard for scientific computing programming languages C, C++ or Fortran and parallel programming technologies MPI and OpenMP.

In any case, one should start with optimization for modern CPUs mainly focusing on scaling efficiency and vectorization. In most cases this optimization is beneficial for Xeon Phi as well. An existing code can be ported to Xeon Phi by just recompiling it with the Intel Compiler; this is a substantial advantage of Xeon Phi over GPUs that generally require significant modification of the code. In our experience this “no-effort” porting usually requires several minutes in an ideal case to several hours, mostly spent on rebuilding third-party libraries. Performance of such a porting can be discouraging even for codes that are quite efficient on CPUs. It might be helpful to use profiling tools to discover the most time-consuming routines and performance-limiting factors. Pieces of code that were fast enough not to be optimized and parallelized on CPU could become a bottleneck for Xeon Phi due to larger amount of cores and poor single-core performance. After such unexpected bottlenecks are eliminated, one should generally focus on scaling efficiency and vectorization in the most time-consuming routines and loops.

We have employed this approach to porting our implementation of the Particle-in-Cell method, gradually profiling, solving performance issues and measuring the effect of optimization on CPU and Xeon Phi. Only after our optimization resources on CPU were exhausted, we proceeded to trying Xeon Phi-specific optimizations. Those included manual vectorization using intrinsics translated into vector instructions of Xeon Phi, using large memory pages to reduce data translation lookaside buffer miss rate, non-temporal stores for better cache utilization efficiency, and adjusting the number of processors and threads in MPI + OpenMP scheme. Most of the mentioned optimization techniques did not yield a significant, if any, performance benefit for our application, a notable exception is manual vectorization via intrinsics.

## 4. Porting and optimization of the Particle-in-Cell code PICADOR

### 4.1. PICADOR Particle-in-Cell code

PICADOR [23,24] is a fully parallel 3D Particle-in-Cell implementation capable of running on heterogeneous cluster systems with CPUs, GPUs and Xeon Phi coprocessors. The PICADOR features include finite-difference time-domain and numerical dispersion free field solvers, Boris particle pusher, cloud-in-cell (CIC) and triangular-shaped cloud (TSC) particle form factors, Villasenor–Buneman and Esirkepov current deposition, ionization, moving frame, and dynamic load balancing. These numerical schemes form a rather standard set for Particle-in-Cell codes. Each MPI process handles a part of the simulation area (domain) using either a multicore CPU or Xeon Phi coprocessor via OpenMP, or GPU via CUDA. All MPI exchanges occur only between processes handling neighboring domains.

The baseline version was developed taking into account some well-known performance considerations, a short description is given below.

The values of each vector component are stored separately in a 3D array wrapped into 1D. Current values are written directly to global current arrays, similarly, field values are read directly from global field arrays. The key performance consideration is particle storage. We use a separate array of particles for each cell with an Array-of-Structures layout. This approach differs from the widely used global particle array with sorting strategy [14] and is closer to data structures used for GPU-based implementation (but without supercells common for GPUs).

At the most time consuming particle push and current deposition stages particles are processed in a cell-by-cell order. Particles in several cells are processed in parallel using OpenMP in one-pragma style. After each particle push we perform a check and, in case a particle leaves the current cell, update the particle storage structure accordingly. This migration check is done partly in parallel: each thread has its own buffer for migrating particles, after all checks are done all buffers are merged and processed sequentially. At the current deposition stage each thread writes to its own current buffer to avoid data races, all buffers are summed after the stage is over.

### 4.2. Benchmark, hardware and performance measurement details

For all the performance measurements presented in this section we used a frozen plasma benchmark with  $40 \times 40 \times 40$  grid, 50 particles per cell, and 1000 time steps. We used a first-order CIC particle form factor for field interpolation and current deposition, and performed all calculations in double precision. The time given in the tables refers only to the computational phase which is a sum of particle push, current deposition and field update.

Computational experiments were conducted on a node of the Lobachevsky cluster system at the University of Nizhni Novgorod with an 8-core Intel Sandy Bridge E5-2660 CPUs (2.2 GHz), Hyperthreading enabled, 64 GB RAM, and 2 Intel Xeon Phi 5110P coprocessors, each with 60 cores and 8 GB RAM. The peak performance of each CPU in double precision is 140 GFLOPS, and the peak performance of Intel Xeon Phi 5110P is 1 TFLOPS. The code was compiled with Intel C++ Compiler 15.0.3 and Intel MPI 4.1.

The configuration of MPI processes per device and OpenMP threads per core can significantly influence performance on both CPU and Xeon Phi, the optimal configuration generally depends on many factors [19,31]. We tried multiple configurations and the empirically optimal choice for all versions of the code described in this section was to run a single MPI process per device (CPU or Xeon Phi) with 2 OpenMP threads per core on CPU (Hyperthreading), 4 threads per core on Xeon Phi, and thread affinity set to scattered.

**Table 1**

Performance of the baseline version on CPU and Xeon Phi.

| Stage              | Time [s] |          |
|--------------------|----------|----------|
|                    | CPU      | Xeon Phi |
| Particle push      | 163.1    | 134.8    |
| Current deposition | 61.3     | 81.3     |
| Field update       | 0.8      | 7.7      |
| Total              | 225.2    | 222.8    |

**Table 2**

Performance of the version with improved memory locality.

| Stage              | Time [s] |          | Speedup to the baseline |          |
|--------------------|----------|----------|-------------------------|----------|
|                    | CPU      | Xeon Phi | CPU                     | Xeon Phi |
| Particle push      | 56.9     | 41.3     | 2.87×                   | 3.26×    |
| Current deposition | 14.0     | 16.9     | 4.38×                   | 4.81×    |
| Field update       | 0.8      | 7.7      | 1.00×                   | 1.00×    |
| Total              | 71.7     | 65.9     | 3.14×                   | 3.38×    |

A comparison of configurations for the final version of the code on Xeon Phi is given in the end of this section.

Table 1 presents results of performance of the baseline version on the CPU and Xeon Phi. On Xeon Phi we used native mode and porting required only rebuilding the code and libraries with compiler options for Xeon Phi support (-mmic). The performance on Xeon Phi is very close to CPU, but the time distribution between stages is different with a faster particle push and a significantly slower field update. Thus, a no effort port gives reasonable performance (taking into account that it took only several hours) but further optimization is required.

### 4.3. Improving memory locality

A natural and widely used idea for efficient implementation of the Particle-in-Cell method is to use the physical locality of the method – each particle is interacting only with several closest grid values – and transform it into memory locality to allow cache-friendly implementation. We use a CIC particle form factor and Yee grid, in this case the particles in each cell interact only with the 27 closest grid values for each field and current component (a cube with side 3).

Before processing the particles of a cell we preload the corresponding 27 surrounding field values into a small local array and use these values for the field interpolation. In a similar way, we accumulate currents created by particles of a cell in a small local array and add it to the global array after all particles are processed. Thus, we replace the majority of memory operations with global field and current arrays by the same operations on local arrays. A similar approach can be applied for other particle form factors and grids with higher order form factors requiring larger local arrays.

The performance of this version and of the baseline version is compared in Table 2. Improving memory locality yields over 3× benefit over the baseline on both CPU and Xeon Phi.

### 4.4. Enhancing scalability on shared memory

The efficiency of scaling on shared memory is important for multicore CPUs and even more so for Xeon Phi. In the baseline version for each thread there was a separate copy of all current values to avoid data races during parallel current deposition with reduction of those copies performed afterwards. Memory overuse due to maintaining the copies and performance overhead on reduction are tolerable for 16 threads on CPUs, but not for 240 threads on Xeon Phi. Thus, we implemented another parallel current deposition scheme [30] that does not replicate all current values. Again, we employed the locality properties described in the

**Table 3**

Performance of the version with improved memory locality and enhanced scalability.

| Stage              | Time [s] |          | Speedup to the baseline |          |
|--------------------|----------|----------|-------------------------|----------|
|                    | CPU      | Xeon Phi | CPU                     | Xeon Phi |
| Particle push      | 52.1     | 37.1     | 3.13×                   | 3.63×    |
| Current deposition | 13.9     | 13.1     | 4.41×                   | 6.21×    |
| Field update       | 0.7      | 1.8      | 1.14×                   | 4.28×    |
| Total              | 66.7     | 52.0     | 3.38×                   | 4.28×    |

previous subsection. Since the particles in each cell only contribute to the grid values in a  $3 \times 3 \times 3$  surrounding cube, particles that are distant enough from one another can be processed in parallel without any risk of data races. Namely, for the CIC form factor we can concurrently process cells that have 2 unprocessed cells in between; the same idea can be applied to other form factors (probably with a larger surrounding cube). Thus, the current deposition consists of 27 particle traversals in checkerboard order, each internal traversal is parallel, only with one synchronization at the end of the traversal.

Next we eliminated the sequential migration of particles between cells. Due to the relation between the space and time steps, a particle cannot pass a distance greater than the cell size in one time step. Thus, a migrating particle is necessarily located in a neighbor cell. We again applied the checkerboard order parallelization scheme [30]. For each cell we created a buffer for the particles migrating to this cell. After pushing a particle each thread computes a new cell index and in case of migration writes the particle to a buffer of the new cell. Because of locality and checkerboard traversal order there is no need for synchronization except at the end of each traversal.

The performance of this version is presented in Table 3. Improving scalability resulted in  $1.07\times$  speedup on CPU and  $1.27\times$  speedup on Xeon Phi, strong scaling efficiency on shared memory grew from 92.7% to 99.2% on CPU and from 61.5% to 78.1% on Xeon Phi (scaling efficiency was computed relative to the single core case with the same number of threads per core).

#### 4.5. Improving vectorization

Efficient vectorization is a key factor in achieving good performance on CPUs and particularly on Xeon Phi coprocessors. The main reason for lackluster performance on Xeon Phi is poor vectorization of the code.

First we tried changing the particle data layout from array-of-structures to structure-of-arrays that is generally beneficial towards vectorization. However, this did not change performance as the main hotspots of the code, field interpolation and current deposition, mostly depend on the grid data layout.

We tried two approaches to vectorization. First we tried to assist the compiler with auto-vectorization by using special directives of Intel Compiler (such as `#pragma ivdep` and `#pragma simd`) and loop splitting. It allowed us to vectorize the implementation of the field update and the Boris method for the particle push, but it did not vectorize the more time-consuming field interpolation and current deposition because of the complicated memory access pattern. Thus, the vectorization led to a modest speedup of the particle push and field update, as shown in Table 4.

The main reason for lackluster speedup due to vectorization is that for each cell we have to preload and store  $3 \times 3 \times 3$  arrays of field and current density components, while each particle uses a  $2 \times 2 \times 2$  subarray depending on its position inside the cell. The implementation uses indirect indexing which renders vectorization of the loop over particles inefficient, particularly on Xeon Phi.

To eliminate indirect indexing the field and current density values used for particles of a cell have been repacked into eight

**Table 4**

Performance on CPU and Xeon Phi after improving vectorization by means of assisting compiler auto-vectorization.

| Stage              | Time [s] |          | Speedup to the baseline |          |
|--------------------|----------|----------|-------------------------|----------|
|                    | CPU      | Xeon Phi | CPU                     | Xeon Phi |
| Particle push      | 45.6     | 20.2     | 3.58×                   | 6.67×    |
| Current deposition | 13.8     | 13.0     | 4.44×                   | 6.25×    |
| Field update       | 0.6      | 0.8      | 1.33×                   | 9.63×    |
| Total              | 60.0     | 34.0     | 3.75×                   | 6.55×    |

**Table 5**

Performance on CPU and Xeon Phi after improving vectorization with intrinsic functions.

| Stage              | Time [s] |          | Speedup to the baseline |          |
|--------------------|----------|----------|-------------------------|----------|
|                    | CPU      | Xeon Phi | CPU                     | Xeon Phi |
| Particle push      | 40.6     | 18.8     | 4.02×                   | 7.17×    |
| Current deposition | 12.9     | 10.1     | 4.75×                   | 8.05×    |
| Field update       | 0.6      | 0.8      | 1.33×                   | 9.63×    |
| Total              | 54.1     | 29.7     | 4.16×                   | 7.50×    |

$2 \times 2 \times 2$  arrays, each corresponding to an octant of a cell. For each element of these arrays the six components of the electromagnetic field are packed into one 512-bit vector register on Xeon Phi or two 256-bit registers on CPU, in both cases effectively using 75% of vector capacity. The three components of the current density are processed in the same fashion occupying one vector register on both CPU and Xeon Phi, effectively using 75% and 37.5% of vector capacity, respectively. For each particle we determine the octant it belongs to and perform the corresponding operation with vector registers using intrinsic functions operating on all components of the field or current density. The scheme is shown in Fig. 2. Compared to the auto-vectorized version, this modification yields an additional  $1.11\times$  speedup on CPU and  $1.15\times$  speedup on Xeon Phi, as shown in Table 5.

Overall, improving the vectorization resulted in  $1.23\times$  speedup on CPU and  $1.75\times$  speedup on Xeon Phi compared to the previous, partially vectorized version. The main obstacle for efficient vectorization is indirect addressing caused by usage of the standard Yee grid for values of the electro-magnetic field and current density [26]. As shown in [22], vectorization can be done much easier for a more straightforward grid.

The strong scaling efficiency of the final version on shared memory is 99.1% on CPU and from 77.6% on Xeon Phi. We also ran the final version of the code on Xeon Phi using several other configurations of processes and threads instead of a single MPI process with 4 threads per core. Using a single process with 1, 2 and 3 threads per core (thread affinity set to scattered) results, respectively, in  $1.92\times$ ,  $1.28\times$  and  $1.15\times$  slowdowns compared to the configuration with 4 threads per core. Then we fixed the total number of threads and tried using several MPI processes (2, 4, 6, 10, 12) choosing the number of OpenMP threads so that there were in total 240 threads on Xeon Phi and tried different thread affinity. The performance of these configurations was very close to the original one with the slowdowns between  $1.01\times$  and  $1.05\times$ . Therefore, the distribution of threads between processes does not significantly affect the performance of our code as long as all the hardware threads on Xeon Phi are used.

## 5. Performance evaluation for a real physical problem

### 5.1. Problem statement

In this section we present the results of efficiency measurements of our Xeon Phi implementation while solving a real physical problem. As an example, we chose laser ion acceleration in the so-called Target Normal Sheath Acceleration (TNSA) regime [33]. In

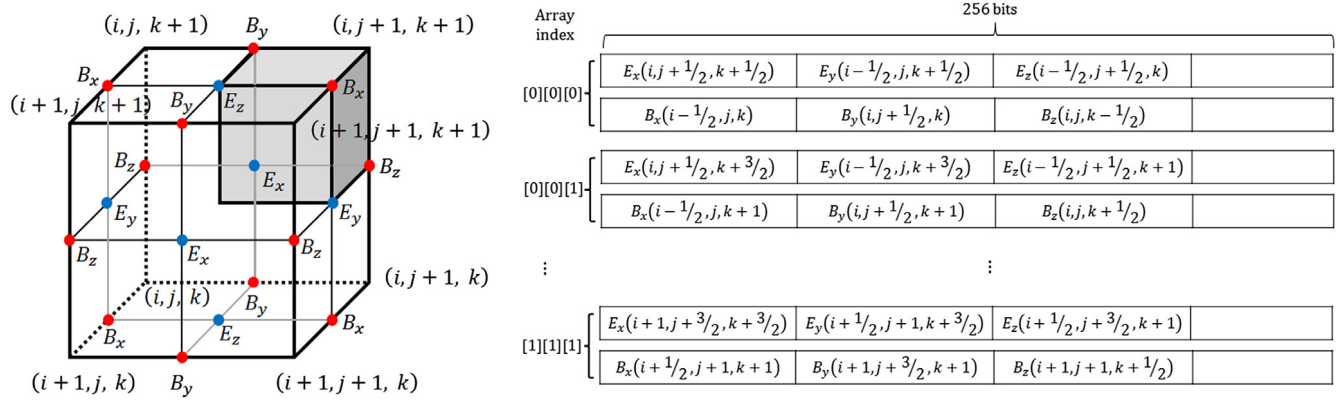


**Fig. 2.** Scheme of vectorization of field interpolation with intrinsic functions on CPU. Left: a cell of Yee grid with one octant selected. Right: contents of the  $2 \times 2 \times 2$  array used for field interpolation for particles of the selected octant. The scheme on Xeon Phi is the same with one 512-bit register instead of two 256-bit registers.

this regime ions are accelerated from the rear side of a thin (sub-micron thick) solid-state target by a quasistatic electron sheath. This sheath is formed by hot electrons accelerated to relativistic energies by a laser field at the front surface of the target. One of the main problems of the scheme is its low efficiency [5]. Recently, it has been suggested to use a surface grating on the irradiated side of the target to increase the laser–electron coupling and therefore increase the amount of energy transferred from the laser radiation to the ions [34]. This proposal has been tested via a number of 2D numerical simulations [35–37], however for more realistic results full 3D simulations are needed especially for the investigation of complex gratings. Realistic 3D simulations, however, are known to be extremely resource-demanding and thus may greatly benefit from the opportunity to be run on heterogeneous systems utilizing the resources of Xeon Phi coprocessors.

We investigated the irradiation of a  $0.3 \mu\text{m}$  thick target composed of  $\text{Au}_{197}^{+31}$  ions with plasma density corresponding to electron concentration  $3 \times 10^{21} \text{ cm}^{-3}$  (overdense parameter  $n_0 = 30$ ). The accelerated proton layer with the same electron concentration and  $0.1 \mu\text{m}$  thickness has been attached to rear side of the target. The grating in the form of a rectangular brush modulated along the  $y$ -axis only has been placed on the irradiated side. The grating height has been chosen to be equal to  $0.3 \mu\text{m}$ , the thickness of a single element was equal to  $0.15 \mu\text{m}$  and the grating period was equal to  $0.5 \mu\text{m}$ . The initial temperature of all plasma components was 100 eV which is similar to values usually expected from collisional heating.

The laser pulse normally impinged on the target was taken to be a plane wave infinite in transversal direction, propagating along the  $x$ -axis and linearly polarized in the  $y$ -direction. It had a Gaussian envelope along the propagation axis with duration at full width half maximum equal to 42 fs and its wavelength equal to  $1 \mu\text{m}$ . The laser intensity at the pulse maximum reached  $3.75 \times 10^{19} \text{ W/cm}^2$  (dimensionless amplitude  $a_0 = 5.2$ ). These parameters are typical for Ti:Sa terawatt systems widespread nowadays in laboratories worldwide.

The simulation area was a box  $12 \times 1 \times 1 \mu\text{m}$  in size covered with a  $512 \times 64 \times 64$  grid. The time step was equal to 0.026 fs and the total simulation time was 300 fs, requiring 11512 time steps to complete.

The maximal ion energy  $W_{\text{max}}$  is sensitive to the number of macroparticles per cell  $N_{\text{PPC}}$ . This is due to the fact that a higher  $N_{\text{PPC}}$  provides a better resolution of the tail of the sheath electron energy distribution and this tail is known to define the maximal energy  $W_{\text{max}}$  reached by the accelerated ions. In our simulations we varied the PPC-parameter within the 10–150 range to investigate the dependence of  $W_{\text{max}}$  on  $N_{\text{PPC}}$ , which is shown in Fig. 3. One can see that the obtained maximal ion energy steadily grows with increasing  $N_{\text{PPC}}$  until it reaches 14.8 MeV at  $N_{\text{PPC}} = 80$  and does not grow further. This feature of the considered problem makes

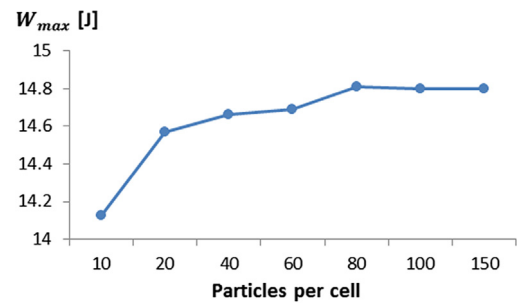


**Fig. 3.** Dependence of the maximal ion energy on the number of particles per cell.

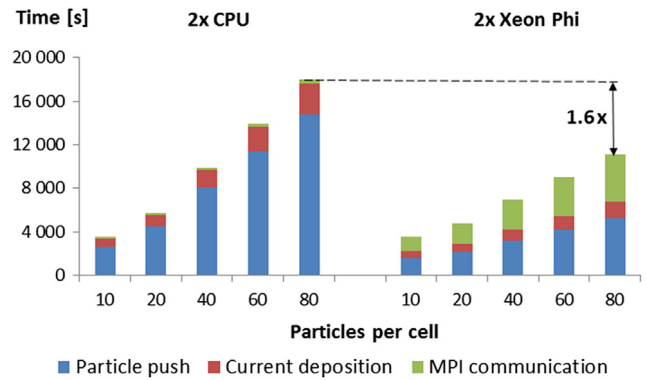


**Fig. 4.** Performance of 2× CPUs and 2× Xeon Phi coprocessors on a TNSA simulation with 10–80 macroparticles per cell.

it highly favorable for running on Xeon Phi coprocessors because they perform simulations with high  $N_{\text{PPC}}$  faster than CPU.

## 5.2. Performance evaluation

We ran the simulation on 2 CPUs and 2 Xeon Phi coprocessors with the number of particles per cell varying from 10 to 80. In both cases 1 MPI process per device was used with 1 OpenMP thread per core on CPU and 4 threads per core on Xeon Phi. The results are presented in Fig. 4. The three most time-consuming stages are the particle push, the current deposition and the MPI exchanges. Other stages, including the field solver, pulse generation and absorbing boundary conditions took negligible time.

The MPI communication phase on Xeon Phi takes a much longer time, not only due to the additional data transfer between the coprocessor and host occurring with each MPI data exchange, but mainly due to the preparation of the data to be transferred involving the copying of the boundary of 3D arrays to 1D arrays,

that is not fully parallelized. The total computational time on CPU and Xeon Phi is close for  $N_{ppc} = 10$  with Xeon Phi steadily outperforming CPU as the  $N_{ppc}$  grows. For  $N_{ppc} = 80$ , simulation on Xeon Phi is  $1.6\times$  times faster compared to CPU with  $2.8\times$  speedup of particle push,  $2.0\times$  speedup of current deposition,  $2.6\times$  overall speedup of computational core, and significantly slower data exchanges and related operations.

## 6. Summary

This paper studies the potential of using Intel Xeon Phi coprocessors as accelerators for Particle-in-Cell plasma simulations. We discuss the Xeon Phi features influencing the performance of Particle-in-Cell implementations. The limitation of 16 GB RAM on Xeon Phi is not a severe constraint for Particle-in-Cell simulations. Due to the massive parallelism potential of the Particle-in-Cell method it can efficiently utilize Xeon Phi. We found that an important limiting factor is the low single-core performance, particularly for code that is not ideally vectorized. In practice, this means that the pieces of code fast enough not to be parallelized or vectorized on CPU can become a bottleneck for Xeon Phi and therefore have to be rewritten.

We support these conclusions by porting the existing 3D Particle-in-Cell plasma simulation code PICADOR to Xeon Phi coprocessors. The parallel C++ code using MPI and OpenMP was originally ported by means of just recompiling with performance on Xeon Phi close to that of an 8-core CPU. We demonstrate a step-by-step application of standard optimization techniques: improving memory locality, scaling efficiency and vectorization that leads to an overall speedup of  $4.2\times$  on CPU and  $7.5\times$  on Xeon Phi. In terms of programming effort, the modifications concerned less than 3% of the source code lines and took about a month including developing, debugging, testing and profiling, taking into account that some ideas and implementations proved to be inefficient.

In the test problem with 50 macroparticles per cell, the final version achieves 16.9 ns per particle update on CPU and 9.3 ns per particle update on Xeon Phi in double precision. In the simulation of laser ion acceleration, two Xeon Phi coprocessors outperform two CPUs by a factor of 1.6. Our implementation demonstrates good scaling on shared memory with 240 threads of Xeon Phi. The most challenging aspect of efficient implementation is the vectorization of the field interpolation and the current deposition due to non-unit stride memory access. Overall, the Particle-in-Cell method, although not ideally suited for Xeon Phi coprocessors due to vectorization issues, can achieve a significant performance gain on modern heterogeneous clusters. To have a primary assessment of the scaling potential on several Xeon Phi coprocessors we ran a simulation of laser wakefield acceleration with a  $512 \times 512 \times 512$  grid and 8 particles per cell on the RSC PetaScale system with 64 Intel Xeon Phi 7120D coprocessors. We obtained a  $3.6\times$  speedup on 64 coprocessors compared to 16 coprocessors (the smallest number to fit the simulation), that corresponds to a 90% strong scaling efficiency.

In terms of further development, the second-generation Intel Knights Landing coprocessors have significant advances as compared to the current Xeon Phi architecture, including a notable improvement of the single-core performance and support for out-of-order execution, and capability for direct data exchanges between coprocessors avoiding PCI Express. Higher single-core performance will allow efficient Particle-in-Cell simulation with a lower number of macroparticles per cell and alleviate, to a certain extent, performance degradation of a poorly vectorized or parallelized code. The capability for coprocessor-only simulation with direct data transfers provides good scaling potential. These improvements together with our good experience of porting PICADOR to existing hardware make Xeon Phi coprocessors a very promising hardware platform for high-performance Particle-in-Cell plasma simulation.

## Acknowledgments

This study was partially supported by the RFBR, research project Nos. 14-07-31211 and 15-37-21015 and by the grant (the agreement No. 02.B.49.21.0003 dated August 27, 2013 between The Ministry of Education and Science of the Russian Federation and the Lobachevsky State University of Nizhni Novgorod). The authors are grateful to Dr. Christopher Harvey of Chalmers University of Technology, Sweden, for his help with editing.

## References

- [1] G. Mourou, T. Tajima, S. Bulanov, *Rev. Modern Phys.* 78 (2) (2006) 309–371.
- [2] A.V. Korzhimanov, A.A. Gonoskov, E.A. Khazanov, A.M. Sergeev, *Phys.-Usp.* 54 (1) (2011) 9–28.
- [3] I.Yu. Kostyukov, A.M. Pukhov, *Phys.-Usp.* 58 (1) (2015) 81–88.
- [4] A. Macchi, M. Borghesi, M. Passoni, *Rev. Modern Phys.* 85 (2) (2013) 751–793.
- [5] V.Y. Bychenkov, A.V. Brantov, E.A. Govras, V.F. Kovalev, *Phys.-Usp.* 58 (1) (2015) 71–81.
- [6] U. Teubner, P. Gibbon, *Rev. Modern Phys.* 81 (2) (2009) 445–479.
- [7] A. Di Piazza, C. Müller, K.Z. Hatsagortsyan, C.H. Keitel, *Rev. Modern Phys.* 84 (3) (2012) 1177–1228.
- [8] N.B. Narozhny, A.M. Fedotov, *Phys.-Usp.* 58 (1) (2015) 95–102.
- [9] M. Ghoranneviss, A.S. Elahi, *Sci. Technol. Nucl. Ins.* 2014 (2014) 802054.
- [10] K.W.D. Ledingham, P.R. Bolton, N. Shikazono, C.-M. Ma, *arXiv*: 1405.2657.
- [11] F. Albert, A.G.R. Thomas, S.P.D. Mangles, S. Banerjee, S. Corde, A. Flacco, M. Litos, D. Neely, J. Vieira, Z. Najmudin, R. Bingham, C. Joshi, T. Katsouleas, *Plasma Phys. Control. Fusion* 56 (8) (2014) 084015.
- [12] C.K. Birdsall, A.B. Langdon, *Plasma Physics via Computer Simulation*, CRC Press, 2004.
- [13] M. Shoucri, *Commun. Nonlinear Sci. Numer. Simul.* 13 (1) (2008) 174–182.
- [14] R.A. Fonseca, J. Vieira, F. Fiuza, A. Davidson, F.S. Tsung, W.B. Mori, L.O. Silva, *Plasma Phys. Control. Fusion* 55 (12) (2013) 124011.
- [15] K.J. Bowers, B.J. Albright, L. Yin, B. Bergen, T.J.T. Kwan, *Phys. Plasmas* 15 (2008) 055703.
- [16] A. Pukhov, *J. Plasma Phys.* 61 (1999) 425–433.
- [17] J.-L. Vay, D.L. Bruhwiler, C.G.R. Geddes, W.M. Fawley, S.F. Martins, J.R. Cary, E. Cormier-Michel, B. Cowan, R.A. Fonseca, M.A. Furman, W. Lu, W.B. Mori, L.O. Silva, *J. Phys.: Conf. Ser.* 180 (2009) 012006.
- [18] H. Burau, R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T.E. Cowan, R. Sauerbrey, M. Bussmann, *IEEE Trans. Plasma Sci.* 38 (10) (2010) 2831–2839.
- [19] J. Jeffers, J. Reinders, *High Performance Parallelism Pearls*, Morgan Kaufmann, 2014.
- [20] I.M. Kulikov, I.G. Chernykh, A.V. Snytnikov, B.M. Glinskiy, A.V. Tutukov, *Comput. Phys. Comm.* 186 (2015) 71–80.
- [21] J. Dongarra, M. Gates, A. Haidar, Y. Jia, K. Kabir, P. Luszczek, S. Tomov, *Lecture Notes in Comput. Sci.* 8384 (2014) 571–581.
- [22] H. Nakashima, *Comput. Electr. Eng.* 46 (2015) 81–94.
- [23] S. Bastrakov, R. Donchenko, A. Gonoskov, E. Efimenko, A. Malyshev, I. Meyerov, I. Surmin, *J. Comput. Sci.* 3 (2012) 474–479.
- [24] S. Bastrakov, I. Meyerov, I. Surmin, E. Efimenko, A. Gonoskov, A. Malyshev, M. Shiryayev, *Lecture Notes in Comput. Sci.* 8488 (2014) 513–514.
- [25] K. Yee, *IEEE Trans. Antennas and Propagation* 14 (3) (1966) 302–307.
- [26] A. Taflov, *Computational Electrodynamics: The Finite-difference Time-domain Method*, Artech House, London, 1995.
- [27] J.P. Boris, in: J.P. Boris, R.A. Shanny (Eds.), *Proceedings of the 4th Conference on Numerical Simulation of Plasmas*, Naval Res. Lab., Washington, D.C., 1970, pp. 3–67.
- [28] H. Qin, S. Zhang, J. Xiao, J. Liu, Y. Sun, W.M. Tang, *Phys. Plasmas* 20 (5) (2013) 084503.
- [29] A. Gonoskov, S. Bastrakov, E. Efimenko, A. Ilderton, M. Marklund, I. Meyerov, A. Muraviev, I. Surmin, E. Wallin, *Phys. Rev. E* 92 (2) (2015) 023305.
- [30] X. Kong, M.C. Huang, C. Ren, V.K. Decyk, *J. Comput. Phys.* 230 (2011) 1676–1685.
- [31] J. Jeffers, J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*, Morgan Kaufmann, 2013.
- [32] W.B. Mori, V.K. Decyk, F.S. Tsung, *Final Report for Phase 1 Enhanced Intellectual Services – Direct PRAC Support – Petascale plasma physics simulations using PIC codes*. <https://bluewaters.ncsa.illinois.edu/documents/10157/11c334f5-353f-4111-8f82-2295b7a28d65>.
- [33] S.C. Wilks, A.B. Langdon, T.E. Cowan, M. Roth, M. Singh, S. Hatchett, M.H. Key, D. Pennington, A. MacKinnon, R.A. Snavely, *Phys. Plasmas* 8 (2) (2001) 542–549.
- [34] Y. Nodera, S. Kawata, N. Onuma, J. Limpouch, O. Klimov, T. Kikuchi, *Phys. Rev. E* 78 (4) (2008) 046401.
- [35] K. Takahashi, S. Kawata, D. Satoh, Y.Y. Ma, D. Barada, Q. Kong, P.X. Wang, *Phys. Plasmas* 17 (9) (2010) 093102.
- [36] K.H. Pae, I.W. Choi, S.J. Hahn, J.R. Cary, J. Lee, *Phys. Plasmas* 16 (7) (2009) 073106.
- [37] A. Andreev, N. Kumar, K. Platonov, A. Pukhov, *Phys. Plasmas* 18 (10) (2011) 103103.