Commun. Comput. Phys.
doi: 10.4208/cicp.070813.280214a

Vol. **16**, No. 3, pp. 599-611
September 2014

Parallelization of an Implicit Algorithm for Multi-Dimensional Particle-in-Cell Simulations

George M. Petrov* and Jack Davis

Naval Research Laboratory, Plasma Physics Division, 4555 Overlook Ave. SW, Washington, DC 20375, USA.

Received 7 August 2013; Accepted (in revised version) 28 February 2014

Available online 24 June 2014

Abstract. The implicit 2D3V particle-in-cell (PIC) code developed to study the interaction of ultrashort pulse lasers with matter [G. M. Petrov and J. Davis, Computer Phys. Comm. 179, 868 (2008); Phys. Plasmas 18, 073102 (2011)] has been parallelized using MPI (Message Passing Interface). The parallelization strategy is optimized for a small number of computer cores, up to about 64. Details on the algorithm implementation are given with emphasis on code optimization by overlapping computations with communications. Performance evaluation for 1D domain decomposition has been made on a small Linux cluster with 64 computer cores for two typical regimes of PIC operation: "particle dominated", for which the bulk of the computation time is spent on pushing particles, and "field dominated", for which computing the fields is prevalent. For a small number of computer cores, less than 32, the MPI implementation offers a significant numerical speed-up. In the "particle dominated" regime it is close to the maximum theoretical one, while in the "field dominated" regime it is about 75-80 % of the maximum speed-up. For a number of cores exceeding 32, performance degradation takes place as a result of the adopted 1D domain decomposition. The code parallelization will allow future implementation of atomic physics and extension to three dimensions.

PACS: 52.38.-r, 52.38.Ph, 52.50.Jm

Key words: Particle-in-cell, Maxwell equations, MPI, lasertarget interaction.

1 Introduction

The particle-in-cell (PIC) codes are ubiquitous and have many applications covering diverse scientific areas such as astrophysics, plasma physics, microelectronics and chemistry [1–3]. PIC codes are also at the forefront of simulation tools for modeling laser-matter interactions since they can adequately model both the laser radiation and the response of the material allowing a self-consistent description of particles and fields. One

*Corresponding author. *Email addresses:* george.petrov@nrl.navy.mil (G. M. Petrov), jack.davis@nrl.navy.mil (J. Davis)

example is the interaction of short-pulse lasers with thin foils, which are routinely used in laboratories around the world for particle acceleration, x-ray generation, and other scientific endeavors such as laser nuclear physics.

The modeling of short-pulse lasers interacting with thin foils faces a host of challenges. Typically, the electromagnetic fields of the laser exceed the atomic field strength and the material becomes instantaneously ionized by Optical Field Ionization. The resulting dense plasma may be opaque for the laser radiation and the material starts to act as a "mirror". The electromagnetic fields can not penetrate the plasma they created and decay inside the material on a scale length of only 10-50 nm, which can be up to three orders of magnitude smaller than the thickness of the foil [4]. In addition, the spatial distribution of the material is highly non-uniform: the target is very dense and concentrated in a narrow region of the computational domain (see Fig. 1), contrary to other problems in which the plasma is distributed uniformly. As a result, the numerical solution of the problem requires high spatial resolution and the number of particles is often prohibitively large. The problem is pushed to the extreme in virtually every numerical aspect, which makes the simulations computationally intensive and numerical algorithms challenging. PIC codes can run for days, sometimes weeks, which is a great impetus for development of efficient and robust PIC codes.

In terms of numerical approach, PIC codes fall into two general categories: explicit and implicit. In explicit PIC information (particle positions and fields) is used only from previous time levels for physical quantities that are already calculated, which makes it straightforward and computationally efficient (per time step). All quantities, electromagnetic field components, current density and particle push are arranged in space and time in the most optimal way and are advanced sequentially in a single sweep. It is, however, subject to severe numerical stability constraints [1–3, 5]. In contrast, implicit PIC codes include information from the next time level, which is more involved and requires some extra logistic and programming efforts, but the payoff can be substantial: the numerical stability improves dramatically, and the number of particles, temporal and spatial resolution requirements are greatly relaxed. In implicit PIC the computational cost per time step for the particles and field update is higher compared to explicit PIC, but implicit PIC can advance with larger time step, use a computational grid with larger cell size and use smaller number of particles. Implicit PIC codes with application to laser-target interactions first appeared in the early 80's [6, 7]. Since then, numerous codes have been developed over the years: ANTHEM [8], AVANTI [9, 10], MACROS [11], DADIPIC [12], OSIRIS [13], LSP [14, 15], CELESTE3D [16], and more recently, iPIC3D [5]. Many of those codes are widely used and are constantly evolving both in terms of applications and numerical implementation. Markidis et al. proposed muti-scale simulations with large dynamic range for studying phenomena spanning over large time scale [5]. Advanced implicit PIC codes have emerged incorporating novel adaptive techniques [17] and adding critical features such as energy conserving schemes [18].

The implicit PIC code, developed at the Naval Research Laboratory (NRL) [19, 20] for studying laser-matter interactions, is powerful enough to handle real-world problems with sufficient accuracy within reasonable period of time (hours to a few days). But in spite of the improvements, the code is still unable to tackle large-scale problems. As an example, if the mass of the target is too large [21], a huge number of particles is required, which is beyond the capabilities of a single processor machine. This problem is unlikely to go away in the near future due to stagnation in CPU speed. The next logical step is code parallelization, i.e. using not one, but many computing cores.

Code parallelization requires careful planning. To achieve maximum efficiency the computational work must be properly distributed among the computer cores overlapping computations with communications. Parallelizing implicit PIC is less straightforward than explicit PIC, because the former is more complex. In explicit PIC all quantities are calculated sequentially in time, while in implicit PIC source terms are predicted or evaluated at a future time level. One of the most challenging aspects of implicit PIC codes stems from the fact that they follow different numerical schemes, which entails different parallelization strategies. For example, iPIC3D solves a second order partial differential equation (PDE) for the electric field components (the wave equation), while we solve two first order PDE's for the electric and magnetic field. In addition, in our code the electric field components are coupled and the corresponding equations must be solved simultaneously (see Section 2). The sequence for pushing particles and computing fields is subject to variations too. In LSP [14, 15] particles are pushed twice per time step and in iPIC3D an iterative procedure for the particle pusher is employed, while in our code the particles are pushed only once. The computational domain decomposition is also different with OSIRIS opting for more advanced domain decomposition schemes. The choice of communications between computer cores is critical for the performance of the (parallelized) code. While some implicit PIC codes have been parallelized using blocking communications [5], we opted for non-blocking communications. Since the parallelization is highly dependent on the specific numerical implementation, the parallelization routine is unique for each approach.

It is the purpose of this paper to present a step-by-step parallelization routine of the implicit PIC code developed in Refs. [19, 20]. We employed the most popular approach for code parallelization, the Message Passing Interface (MPI). The parallelized code is written in C++ using Open MPI [22]. We focus on the parallelization strategy, which is optimized for a relatively small number of computer cores, perhaps up to 64. This choice was dictated by two factors: non-parallelized version of the implicit code is generally more efficient compared to most explicit codes, and parallelization on a massive scale (thousand of computer cores) may require somewhat different approach, as discussed at the end of Section 3. The computational speed-up is evaluated on a small scale Linux cluster system.

2 The implicit PIC method

In this section we briefly reiterate the implicit PIC method developed in Refs. [19,20]. The Maxwell equations

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\varepsilon_0}\left(\vec{\nabla}\times\vec{H}-\vec{j}\right), \tag{2.1a}$$

$$\frac{\partial \vec{H}}{\partial t} = -\frac{1}{\mu_0}\vec{\nabla}\times\vec{E}, \tag{2.1b}$$

with field components $\vec{E}=\left(E_x,E_y,E_z\right)$, $\vec{H}=\left(H_x,H_y,H_z\right)$ and $\vec{B}=\mu_0\vec{H}$ are solved in a Cartesian coordinate system with $\vec{j}=\left(j_x,j_y,j_z\right)$ being the conduction current density. The plasma is modeled by a set of relativistic equations of motion

$$\frac{d\vec{p}_\alpha}{dt} = q_\alpha\left(\vec{E}_\alpha+\frac{\vec{p}_\alpha}{m_\alpha\gamma_\alpha}\times\vec{B}_\alpha\right), \tag{2.2a}$$

$$\tilde{\mathrm{v}}_\alpha = \frac{\vec{p}_\alpha/m_\alpha}{\sqrt{1+\left(\vec{p}_\alpha/m_\alpha c\right)^2}}, \tag{2.2b}$$

$$\frac{d\vec{r}_\alpha}{dt} = \tilde{\mathrm{v}}_\alpha \tag{2.2c}$$

for each computational particle α immersed in an electromagnetic field. In Eq. (2.2), $\vec{r}_\alpha=(x_\alpha,y_\alpha,z_\alpha)$, $\vec{p}_\alpha=\left(p_{\alpha,x},p_{\alpha,y},p_{\alpha,z}\right)$ and $\tilde{\mathrm{v}}_\alpha=\left(\mathrm{v}_{\alpha,x},\mathrm{v}_{\alpha,y},\mathrm{v}_{\alpha,z}\right)$ are the radius vector, particle relativistic momentum and velocity of computational particle α, m_α, q_α and n_α are the mass, charge and the density carried by the particle, respectively, $\gamma_\alpha=\sqrt{1+\left(\vec{p}_\alpha/m_\alpha c\right)^2}$ is the relativistic factor, c is the speed of light and $\vec{E}_\alpha$ and $\vec{B}_\alpha$ are the electric and magnetic field at the particle position $\vec{r}_\alpha$. The Maxwell equations (2.1) are coupled to the particle equations of motion (2.2) through the conduction current density $\vec{j}(\vec{r})=\sum_\alpha\vec{j}_\alpha(\vec{r}_\alpha)W(\vec{r}-\vec{r}_\alpha)$, where $\vec{j}_\alpha=n_\alpha q_\alpha\tilde{\mathrm{v}}_\alpha$ is the current density carried by computational particle α and W is the particle shape function, which is used to distribute quantities from the particle position onto grid nodes and vice versa. The coupling of $\vec{j}$ to the Maxwell equations (2.1) is critical for the performance of the PIC code. In explicit codes the current density is computed directly and the result inserted into Eq. (2.1a). In our implicit algorithm Eq. (2.2a) is inverted and solved for the particle momentum. During the process the electric field is factored out and $\vec{j}(\vec{r})$ is put into the form $\vec{j}(\vec{r})=\sum_\alpha\left(\hat{S}_\alpha\vec{E}_\alpha+\delta\vec{j}_\alpha\right)W(\vec{r}-\vec{r}_\alpha)$, which in vector notations reads:

$$\vec{j}=\hat{S}\vec{E}+\delta\vec{j}. \tag{2.3}$$

In Eq. (2.3) $\widehat{S}$ is a global tensor and $\delta\vec{j}$ is some residual current, both accumulated on grid nodes. Their full derivation is given in Ref. [20]. Inserting (2.3) into the Maxwell equations yields a system of three coupled equations for the electric field components $\vec{E}=\left(E_x,E_y,E_z\right)$:

$$\left(\hat{I}+\hat{S}^{n+1/2}\right)\vec{E}^{n+1}=\left(\hat{I}-\hat{S}^{n+1/2}\right)\vec{E}^{n}+\left(\vec{\nabla}\times\vec{H}^{n+1/2}-\delta\vec{j}\right)\Delta t/\varepsilon_0, \tag{2.4}$$

which are solved simultaneously. The magnetic field components are advanced according to

$$\vec{H}^{n+3/2} = \vec{H}^{n+1/2} - \vec{\nabla} \times \vec{E}^{n+1} \Delta t / \mu_0, \tag{2.5}$$

once the electric field components $\vec{E}^{n+1}$ are calculated. We would like to point out that the arrangement of electric and magnetic fields on the computational cell does not follow the conventional Yee scheme. The electric field is located on grid nodes, while the magnetic field is located in cell center instead. The next section will elaborate on the parallelization of the implicit algorithm.

3 MPI implementation of the implicit algorithm

The code parallelization is based on standard MPI libraries. For maximum efficiency only non-blocking parallel communications among computer cores are used. In MPI the computational work is distributed among many computer cores. Code development is somewhat more complicated since computations flow in parallel on all computer cores simultaneously and need to be well coordinated in order to avoid bottlenecks and delays. Efficient MPI parallelization is achieved by following a few basic rules such as:

- *Load balancing*. The computational load must be approximately equally distributed among the computer cores. Specifically for PIC, the electromagnetic fields are re-computed and the particles are pushed for every computational cycle, therefore, the computer cores must be given equal slice of the computational domain and the particles must be evenly distributed.

- *Overlapping communications with computations*. Since communications are costly, good programming practice requires that some computations are done while data are "in transit" from one processor to another.

Though more sophisticated methods are needed to take full advantage of MPI, we find that these two are sufficient for small scale implementation (tens to hundreds of computer cores). In the following sub-sections the individual steps of the parallelized PIC algorithm are described. They are adapted specifically for our implicit algorithm, but other PIC codes (including explicit) can follow a similar routine with minor modifications.

3.1 Computational domain decomposition

We employ a 2D Cartesian geometry as shown in Fig. 1. The computational domain is defined as $R = \{0 \le x \le L_x, 0 \le y \le L_y\}$ with L_x and L_y being its length and width, respectively. The laser electromagnetic radiation advances from left to right along the "x" axis. At the beginning of the computations the front of the electromagnetic pulse is at the "left" boundary $x = 0$. The foil is perpendicular to the laser beam and has a width

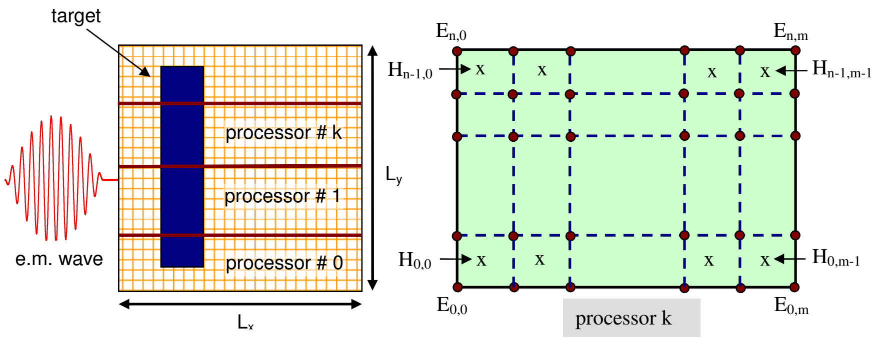

Figure 1: Computational domain, laser radiation, target, and domain decomposition (left). Local computational domain of processor "k" (right). The electric field values are defined on grid nodes, the magnetic field values are defined in cell center. The components of tensor $\widehat{S}$ and vector $\delta\vec{j}$ are also defined on grid nodes.

W and length L. The domain decomposition is shown in Fig. 1. The local computational domains are long narrow strips parallel to the "x" axis. Though there are more efficient ways, such as 2D domain decomposition [22], this partition is simple and load balancing for both particles and fields is accomplished. The boundaries between computer cores are "sharp", i.e. two adjacent computer cores have common grid nodes but no common cells (Fig. 1). Particles are divided among computer cores according to their location and transferred to another core if exiting the computational domain of a local core.

3.2 Initialization

The PIC cycle is preceded by an initialization routine, which includes:

1. Set up a grid on local computer cores according to the space decomposition. We set a "global" grid and assign each computer core a slice of it, forming a local grid with $(m+1)\times(n+1)$ grid points (n is along axis "y" and is approximately equal for all computer cores).

2. Distribute particles among computer cores according to their location and initialize them by assigning (local) coordinates, velocities, charge, etc.

3. Initialize the electromagnetic fields. The electric field components are located on grid nodes, $\vec{E}_{i,j}$, $0\le i\le n$, $0\le j\le m$, while the magnetic fields components $\vec{H}_{i,j}$, $0\le i\le n-1$, $0\le j\le m-1$ are located in cell center [20]. At the beginning of the computations the electromagnetic field components inside the computational domain of every core are set to zero.

3.3 The PIC cycle

The PIC cycle consists of electromagnetic field solver, computation of forces, particle pusher and current density computation followed by interpolation on grid nodes. The

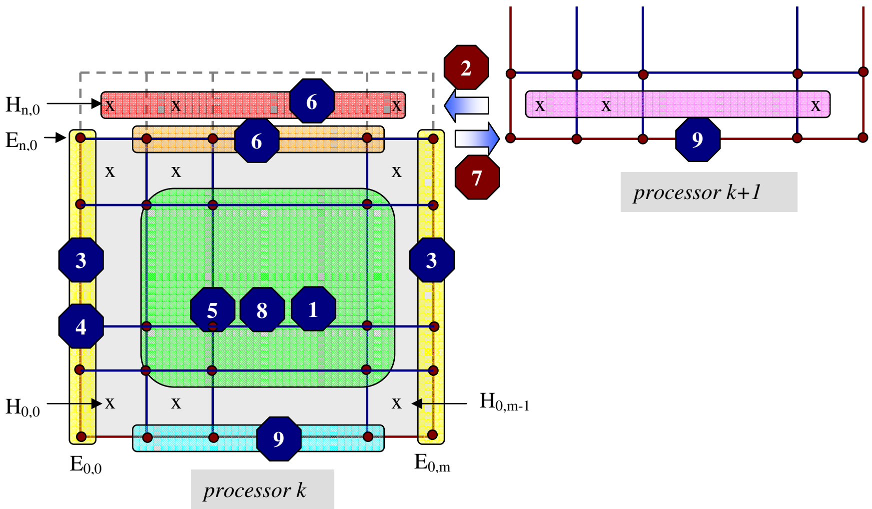

Figure 2: MPI parallelization sequence. The numbers correspond to the steps for computing the electromagnetic fields described in Section 3: (1) compute interior points for tensor $\widehat{S}$ and vector $\delta\vec{j}$; (2) send one row of magnetic field values $\vec{H}_{0,j}$, $0\le j\le m-1$ from processor $k+1$ to processor k; (3) apply boundary conditions at $x=0$ and $x=L_x$ to update $\vec{E}_{i,0}$ and $\vec{E}_{i,m}$, $0\le i\le n$ on processor k; (4) add a source field at $x=0$ to update $\vec{E}_{i,0}$, $0\le i\le n$ on processor k; (5) compute interior points for the electric field on processor k, $\vec{E}_{i,j}$, $1\le i\le n$, $1\le j\le m$; (6) compute the electric field values $\vec{E}_{n,j}$, $0\le j\le m$ belonging to the last row n processor k; (7) send the last row of values $\vec{E}_{n,j}$, $0\le j\le m$ from processor $k-1$ to processor k; (8) compute the magnetic field values $\vec{H}_{i,j}$, $1\le i\le n-1$, $0\le j\le m-1$, for rows $i\ge 1$; (9) compute the magnetic field $\vec{H}_{0,j}$, $0\le j\le m-1$ for row $i=0$.

electromagnetic field solver is broken into nine steps (Fig. 2). The motivation for having so many steps is dictated by the desire to achieve maximum speed and efficiency of the code. The main hurdle is the necessity to exchange information between computer cores, which arises because generally speaking the spatial derivatives of $rot(\vec{E})$ and $rot(\vec{H})$ may require information stored on adjacent computer cores. Since the information exchange is much slower than floating point computations, computational speedup is achieved by carefully overlapping communications with computations. The nine steps, described below, are designed to optimize this process by initiating communications first, performing computations on parts of the computational domain that are independent of the data being sent and finally, receiving the data. For this purpose, we use non-blocking communications between computer cores. The following steps describe the order in which $\vec{E}_{i,j}$ and $\vec{H}_{i,j}$ are computed. The equation number used in describing steps 1-9 refer to that of Ref. [20].

(1) Compute tensor $\widehat{S}$ and vector $\delta\vec{j}$ on grid nodes on each core (Eq. 12). On the boundary between two computer cores the values on grid nodes are incomplete since there is a contribution from particles belonging to the adjacent core. Non-blocking communication is initiated, sending the first row of local values $\widehat{S}_{0,j}$ and $\delta\vec{j}_{0,j}$, $0\le j\le m$ from processor $k+1$ to processor k. Total of

twelve rows are sent, nine for $\widehat{S}$ and three for $\delta\vec{j}$. Later, the data will be received by processor k and added to the last row of local values $\widehat{S}_{n,j}$ and $\delta\vec{j}_{n,j}$, $0\le j\le m$.

(2) In order to compute the electric field on the last row n of processor k, $\vec{E}_{n,j}$, $0\le j\le m$, one needs to know the local values of the magnetic field $\vec{H}_{n,j}$. However, the data for the magnetic field on local processor k span only up to row $n-1$. The needed row n resides on the next processor $k+1$ as row 0. Therefore, we initiate a non-blocking communication by sending the first row of the magnetic field $\vec{H}_{0,j}$, $0\le j\le m-1$ from processor $k+1$ to processor k. Later, the data will be received by processor and stored as local values $\vec{H}_{n,j}$, $0\le j\le m-1$.

(3) Apply boundary conditions at $x=0$ and $x=L_x$ (Eq. 14) to update $\vec{E}_{i,0}$ and $\vec{E}_{i,m}$, $0\le i\le n$ on local processor k.

(4) Add a source field at $x=0$ (Eq. 15) to update $\vec{E}_{i,0}$, $0\le i\le n$ on local processor k.

(5) Compute the interior points for the electric field on local processor k, $\vec{E}_{i,j}$, $1\le i\le n$, $1\le j\le m$ (Eq. 11).

(6) The electric field values $\vec{E}_{n,j}$, $0\le j\le m$ belonging to the last row n are computed next. On the last processor apply boundary condition (Eq. 14). On all other computer cores wait (if necessary) for the communications initiated in steps (1) and (2) to complete, retrieve $\widehat{S}_{n,j}$ and $\delta\vec{j}_{n,j}$, $0\le j\le m$, as well as $\vec{H}_{n,j}$, $0\le j\le m-1$ and compute $\vec{E}_{n,j}$.

(7) The electric field values belonging to the first row $\vec{E}_{0,j}$, $0\le j\le m$ on processor k are still not known, but since for $k\ge 1$ they are identical to the electric field values of the last row n of processor $k-1$ (shared grid nodes), initiate a non-blocking communication by sending the last row of values $\vec{E}_{n,j}$, $0\le j\le m$ computed in step (6) from processor $k-1$ to processor k. Later, the data will be received by processor k and stored as local values $\vec{E}_{0,j}$, $0\le j\le m$. For local processor $k=0$ apply boundary condition (Eq. 14).

(8) While data sent in step (7) are in transit, compute the magnetic field $\vec{H}_{i,j}$, $1\le i\le n-1$, $0\le j\le m-1$, using Eq. 13 for all rows except for the first one, $i=0$. This row can not be computed until the data sent in step (7) are received and become available. Note that no boundary conditions are required for the magnetic field components since all values reside in cell center and are computed using the electric field values on grid nodes.

(9) Wait (if necessary) for communications initiated in step (7) to complete and compute the magnetic field $\vec{H}_{0,j}$, $0\le j\le m-1$ on all computer cores.

The total communication cost is $12m$ for the coefficients forming current density (3), $3(m-1)$ for the magnetic field and $3m$ for the electric field, or the total of $18m$ double precision numbers (eighteen rows), which is about twice the amount of data transfer for explicit PIC codes. However, the overlap of communications with computations compensates to a large extent for the extra communication cost. For example, steps (3)-(5) can be performed while the data sent on steps (1) and (2) are in transit. There is a large amount of data to be computed, on the order of $(m+1)\times n$ grid points (all internal grid nodes plus boundary conditions), without any need of the data currently in transit. The same holds for the magnetic field computation in steps (8) and (9). Overall, the procedure is very efficient and is expected to scale well with the number of computer cores.

The particle pusher is the same as previously described in Ref. [20], but one has to account for particles that cross the boundary with adjacent computer cores. Those that do are put in a separate array, removed from the current core and the array of particles is send to the receiving core. Actually, except for the first and last computer cores, two such arrays are formed, one sent from computer core k to core $k+1$ and the other from core k to core $k-1$. While the data are in transit, more computations can be done such as collisions or ionizations.

A major issue faced in all PIC codes is the interpolation of quantities from grid nodes to the particle position. The particle shape function, W, is a polynomial, often up to fourth order in order to reduce spurious grid heating. The higher the order, the more cells are involved, which presents a problem during parallelization since some of the cells may reside on a neighboring computer core. Using "ghost cells" residing on adjacent computer cores is the most common technique. In the implementation outlined above the use of "ghost cells" is completely avoided since in our implicit PIC linear interpolation yields sufficient accuracy and all quantities are collected within a cell using the procedure outlined in Ref. [3, Eqs. 49-56].

Load balancing is another key issue for optimizing the parallelization process. The numerical grid on which the electromagnetic fields are computed is equidistant in each direction and is therefore evenly distributed in the computational domain. It is intuitively clear that in order to compute the electromagnetic fields efficiently, it suffices to break the computational domain approximately equally among the computer cores. In practice, any type of partition will do, as long as each processor gets an equal piece. However, in laser target interactions the plasma density is strongly non uniform. Initially, all particles are concentrated in a narrow strip and occupy about 1% of the computational domain. As a result, the particles can be a major source of computational imbalance between the computer cores. The 1D decomposition method we chose is the simplest way to achieve computational balance for both particles and fields, though not necessarily the best. Even after the particles move around for a while, each processor still has about equal share of the particles. This choice of domain decomposition is expected to work well for a small number of computer cores only. With increasing the number of computer cores the strips became too narrow and particles start to cross frequently the boundary from one processor to another. The communication cost increases to a point that may not be fully compensated by overlapping communications with computations. Parallelization on a massive scale (thousand of computer cores) may require alternative ways to decompose the computational domain. A 2D decomposition is faster on the on the order of $4/\sqrt{N}$ [22] (N is the number of grid points in each direction), but is logistically a little more complex since each processor has 4 neighbors instead of two. The balance load for the electromagnetic fields is still very good, but not for the particles. In fact, for a 2D decomposition the particles will be initially distributed over a very small number of computer cores since all particles are heavily concentrated in a small region of the computational domain. As the particles move, they would become distributed more evenly over many computer cores. For a small number of computer cores (<100), the 1D decomposition is appropriate, while for a large number of computer cores (>100) we deem the 2D decomposition advantageous.

4 Results and discussions

Our first task was to confirm that any multiprocessor run reproduces the single processor one, i.e. several sets of results from the "original code" were reproduced. Once benchmarking was successfully completed, we evaluated the system speedup using two "standard" simulation runs. The PIC code was run on a single processor and repeated on a small Linux cluster with 64 computer cores. The simulation box is a square with dimensions $\{L_x \times L_y\} = 30\times 30$ μm^2. The number of cells is 1200×1200 and cell size is 25×25 nm^2. Linearly polarized laser pulses with peak intensity $I_0 = 2.5\times 10^{24}$ W/m^2, pulse duration $\tau_{\rm FWHM} = 60$ fs, spot size $D_{\rm FWHM} = 3.3$ μm, and wavelength $\lambda_0 = 1$ μm are used. The two targets are carbon foils with density $\rho = 1000$ kg/m^3, thickness $L_C = 1$ μm (target #1) and $L_C = 0.2$ μm (target #2) and width $W = 28$ μm, with a thin H_2O contamination layer on the back with density $\rho = 1000$ kg/m^3 and thickness $L_{H_2O} = 5$ nm. It is located 3 μm from the "left" boundary. The particles and electromagnetic field components are advanced with a time step $\Delta t = 0.01\lambda_0/c$. The simulations begin at time $t = -10$ fs when the laser pulse enters the computational domain at $x = 0$ and continue for another 180 fs. At the beginning on the simulations the number of computational particles is 2×10^6 for target #1 and 0.4×10^6 for target #2, but during the simulation run it increases due to ionization (Fig. 3).

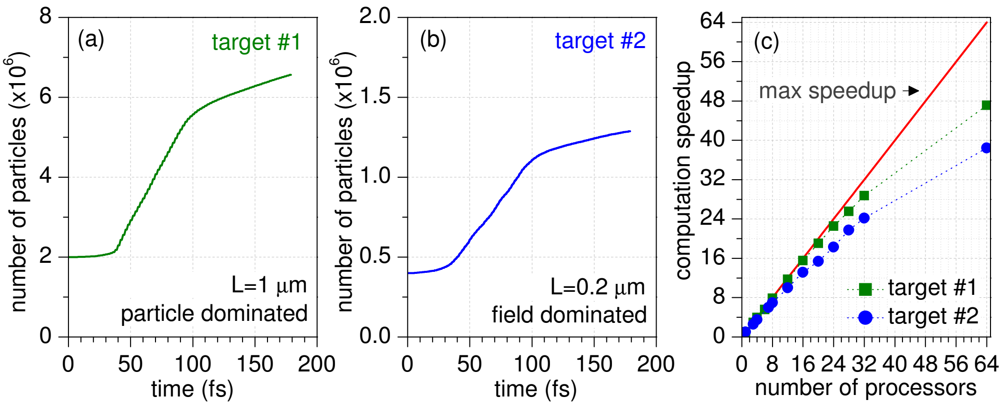

Figure 3: Time evolution of the number of computational particles in the "particle dominated" regime (a), and "field dominated" regime (b). Computation speedup versus number of computer cores for the two regimes (c). Squares: particle dominated regime, cycles: field dominated regime, red dotted line: maximum speedup.

The purpose of choosing two targets is the following. For the thicker target the number of particles is large (several million) and the computations are spent primarily on pushing them. We call this "particle dominated" regime. The thinner target illustrates the opposite, "field dominated" regime, for which most computations are spend on computing the fields. Our goal is to investigate the system speed-up in the two regimes.

Simulations were done on an Intel Xeon 3.16 GHz machine, running 64 bit Red Had Linux 5.4 operating system. Each nodes of the system has a dual Intel Xeon X5460 3.17GHZ quad core processors with 16GB of 667MHZ DDR2 DIMMs and a dual gibabit NICs. OpenMPI version 1.4.1 was used with Intel compiler 11.1. MPI derived data types are built from basic MPI datatypes such as MPI::INT and MPI::DOUBLE. The speed-up calculation was an average of 3 runs for a different number of computer cores. For each run the computation time was tracked and compared to a single-processor run, from which the computation speedup was evaluated. The computation speedup s as a function of number of computer cores N is plotted in Fig. 3c. The red dashed line is the theoretical maximum speed-up. In the "particle dominated" regime, the speed-up is very close to the maximum one. In the "field dominated" regime the speed-up is fairly good, about $\sim$75-80 % of the maximum speed-up, somewhat lower compared to the thicker target. The reason is the increased rate of communications with the number of computer cores, which is more prominent if the number of particles is small. In both cases the speed-up is satisfactory, but we expect that for realistic cases it will be closer to the "particle dominated" regime since the parallelization targets computationally intensive problems, which arise primarily for "thick" targets ($L > 10$ μm) with excessively large number of particles.

The observed speed-up as a function of the number of computer cores falls into two categories. For a modest number of computer cores, $N \leq 32$, it is linear and is close to the maximum speed-up. For $N > 32$ there is a performance degradation and the overall speed-up becomes sub-linear. The performance degradation at large N is a direct consequence of the choice of computational domain decomposition (Fig. 1a). For $N > 32$ the long narrow strips become inefficient as the communication-to-computation cost increases. To see that, consider a grid on a local core with $(m+1)\times n$ grid points. The computation and communication cost scale as $(m+1)\times n$ and $2\times n$ (one for each adjacent processor), respectively. The communication-to-computation cost is therefore $\sim 2/m$. For a small number of computer cores the number of rows m is large enough and communication cost is negligible. With the number of computer cores N increasing $m \sim 1/N$ decreases up to a point where communications start to overwhelm computations. On a more general note, the computation cost is roughly proportional to the surface area given to each core, while the communication cost is proportional to the length of its periphery. Therefore the 2D decomposition has better "surface area to periphery length" ratio compared to the 1D decomposition and is on the order of $4/\sqrt{N}$ faster [22]. For a large number of processors N the 2D decomposition is clearly more advantageous and should be adopted. The 1D decomposition benefits from its simplicity, but it is only efficient on a small scale, with a number of computer cores on the order of 32.

5 Conclusion

The implicit particle-in-cell algorithm developed previously has been parallelized using MPI (Message Passing Interface). The MPI implementation leads to a significant speed-up of the numerical code, especially in the case when computations are dominated by pushing particles. In order to use MPI efficiently, the PIC cycle is split into nine steps, which are arranged so that a reasonably good balancing is achieved and computations overlap with communications whenever possible. The parallelization is optimized for a small number of computer cores, up to 64, with no attempt for parallelization on a massive scale (thousand of computer cores). If the number of computer cores is less than 32, the speed-up is about 75-80 % of the maximum speed-up. However, for a number of cores exceeding 32, performance degradation is observed, which is most likely a result of the simplistic 1D domain decomposition. Future work will focus on inclusion of more detailed atomic physics and a binary collision model for small angle Coulomb scattering, as well as extension to three dimensions.

Acknowledgments

This work was supported by the Naval Research Laboratory under the Base 6.1 program.

References

[1] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, McGraw-Hill, New York, 1981.
[2] C. Birdsall, Particle-in-cell charged-particle simulations plus Monte Carlo Collisions with neutral atoms, PIC-MCC IEEE Trans. Plasma Sci., 19 (1991), 65-85.
[3] J. P. Verboncoeur, Particle simulation of plasmas: review and advances, Plasma Phys. Control. Fusion, 47 (2005), A231-A260.
[4] P. Gibbon, Short Pulse Laser Interactions with Matter, Imperial College Press, London, 2005.
[5] S. Markidis and G. Lapenta, Rizwan-uddin multi-scale simulations of plasma with iPIC3D, Mathematics and Computers in Simulation, 80 (2010), 1509-1519.
[6] R. Mason, Implicit moment particle simulation of plasmas, J. Comp. Phys., 41 (1981), 233-244.
[7] J. U. Brackbill and D. W. Forslund, An implicit method for electromagnetic plasma simulation in two dimensions, J. Comp. Phys., 46 (1982), 271-308.
[8] R. J. Mason, An electromagnetic field algorithm for 2D implicit plasma simulation, J. Comp. Phys., 71 (1987), 429-473.
[9] D. W. Hewett and A. B. Langdon, Electromagnetic direct implicit plasma simulation, J. Comp. Phys., 72 (1987), 121-155.
[10] D. W. Hewett and A. B. Langdon, Recent progress with AVANTI: A 2.5D EM direct implicit PIC code, Computer Phys. Comm., 48 (1988), 127-133.
[11] M. Tanaka, Macroscale implicit electromagnetic particle simulation of magnetized plasmas, J. Comp. Phys., 79 (1988), 209-226.

[12] M. R. Gibbons and D. W. Hewett, The Darwin direct implicit particle-in-cell (DADIPIC) method for simulation of low frequency plasma phenomena, J. Comp. Phys., 120 (1995), 231-247.

[13] R. A. Fonseca, L. O. Silva, F. S. Tsung, V. K. Decyk, W. Lu, C. Ren, W. B. Mori, S. Deng, S. Lee, T. C. Katsouleas and J. C. Adam, OSIRIS: A three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators, Proc. ICCS, Lecture Notes Computer Science, 2331 (2002), 342.

[14] D. R. Welch, D. V. Rose, R. E. Clark, T. C. Genoni and T. P. Hughes, Implementation of an non-iterative implicit electromagnetic field solver for dense plasma simulation, Computer Phys. Comm., 164 (2004), 183-188.

[15] D. R. Welch, D. V. Rose, M. E. Cuneo, R. B. Campbell and T. A. Mehlhorn, Integrated simulation of the generation and transport of proton beams from laser-target interaction, Phys. Plasmas, 13 (2006), 063105.

[16] G. Lapenta, J. U. Brackbill and P. Ricci, Kinetic approach to microscopicmacroscopic coupling in space and laboratory plasmas, Phys. Plasmas, 13 (2006), 055904.

[17] M. E. Innocenti, G. Lapenta, S. Markidis, A. Beck and A. Vapirev, A multi level multi domain method for particle in cell plasma simulations, J. Comp. Phys., 238 (2013), 115-140.

[18] S. Markidis and G. Lapenta, The energy conserving particle-in-cell method, J. Comp. Phys., 230 (2011), 7037-7052.

[19] G. M. Petrov and J. Davis, A two-dimensional electromagnetic field algorithm for high-intensity laser-target interactions, Computer Phys. Comm., 179 (2008), 868-880.

[20] G. M. Petrov and J. Davis, A generalized implicit algorithm for multi-dimensional particle-in-cell simulations in Cartesian geometry, Phys. Plasmas, 18 (2011), 073102.

[21] D. P. Higginson, J. M. McNaney, D. C. Swift, G. M. Petrov, J. Davis, J. A. Frenje, L. C. Jarrott, R. Kodama, K. L. Lancaster, A. J. Mackinnon, H. Nakamura, P. K. Patel, G. Tynan and F. N. Beg, Production of neutrons up to 18 MeV in high-intensity, short-pulse laser matter interactions, Phys. Plasmas, 18 (2011), 100703.

[22] W. Gropp, E. Lusk and A. Skjellum, Using MPI, Portable Parallel Programming with the Message Passing Interface, MIT Press, 1999.