Bull. Nov. Comp. Center, Num. Anal., 17 (2015), 7–15

Parallel template implementation of a Particle-in-Cell code for the simulation of ultrarelativistic beam dynamics*

M.A. Boronina, A.V. Snytnikov

Abstract. In order to simplify the development of high-performance plasma physics codes for hybrid supercomputers, a template implementation of the Particle-in-Cell (PIC) method was created. The template parameters are the problem-specific implementations of "particle" and "cell" (as C++ classes).

Thus, it is possible to develop a PIC code for the supercollider physics problem without studying GPU programming by the beam physicist. Instead, the new physical features are just included into the existing code as new implementations of the "simulation domain" class.

Since the template implementation of the PIC method was created on the basis of the relativistic plasma physics code, the transition to the beam physics is quite simple. Just the two things were changed: initial particle distribution and boundary condition computation. Both changes were performed by means of virtual functions of the "simulation domain" class.

The result is that one single Tesla M2090, GPU computes with approximately the same speed as 24 Intel Xeon cores.

1. Introduction

We present a new parallel particle-in-cell (PIC) algorithm for the numerical modeling of ultrarelativistic beam dynamics in supercolliders. One of the main efficiency parameters of the colliders is luminosity, which describes the total number of events in the interaction cross-section during a certain time. The challenge of modern experiments and collider design projects is to attain high luminosities for the high energy beams. The interaction of the high-energy beams (characterized by the relativistic factor $\gamma \sim 10^3$–10^5) is the limiting factor for the luminosity; however the negative effects can be reduced by the beam configuration optimization.

Numerical modeling is widely used to study the beam-beam effects. A standard approach to the computation of the beam interaction entails the slice rearrangement, where the colliding beams are divided into macro-particles slices with each slice of one beam impacting on the counter-moving beam particles by two-dimensional forces. This approach is commonly used

*Supported by the Russian Science Foundation under Grant 14-11-00485. The programs were written under the RFBR Grants 14-07-00241, 14-01-00392, 15-31-20150, and 14-01-31088.

for cyclic accelerators, where the beam deformations are not strong, the computations of a single interaction are fast and only 5–50 slices and 10^5 macro-particles per beam provide a sufficient accuracy. In the case of a single collision of ultrarelativistic beams of high densities, a strong compression and even disruption ("pinch effect") of a beam may occur, requiring an adequate longitudinal resolution for a number of pinches. The nature of the longitudinal effects is three-dimensional and cannot be simulated by a quasi-3D model. However, in the case of critical beam densities (as in the International Linear Collider projects) the effects may be of considerable importance, the redistribution of energy may lead not only to a low luminosity, but even to a burnt plant. The fine resolution requires an appropriate number of macro-particles in the beam (10^8–10^{10}).

We have developed a new parallel fully 3D algorithm using the Vlasov-Liouville equation for the distribution function of the beam particles and the 3D set of Maxwell's equations. We use the PIC method with the leap-frog scheme, where all the components are calculated on the half-step staggered grids. The scheme allows the calculation of the motion of beams regardless of the collective motion direction when the initial and boundary conditions in the near wave zone are correct. However, the main problem in the computation of the 3D electromagnetic beam fields is a high value of the relativistic factors γ. Comparing the case in question with the case of non-relativistic motion, one can see that the electric field of a relativistic particle *gamma* times increases in the transversal direction and γ^2 times decreases in the longitudinal direction. As usual, the beam field computation for $\gamma \sim 10^3$ requires 10^9 times more operations and makes the computation prohibitive by standard methods even when using supercomputers. To overcome the above difficulties, we have developed a new and efficient method for the calculation of the boundary and initial conditions.

As the beam density distribution is highly nonlinear (Gaussian in each direction) and the beam significantly changes its shape due to the focusing conditions ("hourglass effect"), the majority of the particles is concentrated in a small region. In order to reduce the resulting simulation time, we propose the employment of the GPU-based supercomputers. And since the GPU programming is extremely complex, there is a need for a fast and reliable tool for the transition from the CPU code to the GPU code. Such a tool was proposed in [1].

2. The model description

We consider the motion of the counter charged electron/positron beams in a small rectangular domain $[0, L_x] \times [0, L_y] \times [0, L_z]$. The beams are moving in vacuum in self-consistent electromagnetic fields. The beams are focused by the external focusing field. We assume that the boundaries are located in the

near wave zone, no radiation effects are considered. We need to analyze the particle motion dependence on a given beam configuration[2]. We consider the monoenergetic beam motion strictly along the axis z at $\gamma = 6.85 \cdot 10^3$: The particle density is Gaussian:

$$\rho(x,y,z) = \frac{1}{(2\pi)^{\frac{3}{2}}(\sigma_x^*\sigma_y^*\sigma_z^*)^{\frac{1}{2}}} \exp\left[-\frac{1}{2}\left(\frac{(x-x_c)^2}{\sigma_x^{*2}} + \frac{(y-y_c)^2}{\sigma_y^{*2}} + \frac{(z-z_c)^2}{\sigma_z^{*2}}\right)\right],$$

the transversal momenta are distributed in the crossover plain according to the following law:

$$\rho(X',Y') = \frac{1}{2\pi\sigma_{p_x}^*\sigma_{p_y}^*} \exp\left[-\frac{1}{2}\left(\frac{X'^2}{\sigma_{p_x}^{*2}} + \frac{Y'^2}{\sigma_{p_y}^{*2}}\right)\right],$$

where $\sigma_x^* = \sqrt{\beta_x^*\varepsilon_x^*}$, $\sigma_y^* = \sqrt{\beta_y^*\varepsilon_y^*}$ are the horizontal and vertical beam sizes, $\sigma_{p_x}^* = \sqrt{\varepsilon_x^*/\beta_x^*}$, $\sigma_{p_y}^* = \sqrt{\varepsilon_y^*/\beta_y^*}$ are the corresponding beam divergencies, $\varepsilon_x = \varepsilon_x^* = 5{\cdot}10^{-5}$ and $\varepsilon_y = \varepsilon_y^* = 5{\cdot}10^{-7}$ are the transversal beam emittances, $\beta_x^* = 0.1$ and $\beta_y^* = 0.1$ are the corresponding beta-function values, $\sigma_z = \sigma_z^* = 0.1$ is the beam size along z-axis, the sign * denotes the values in the interaction point. So, the beam size ratio is $\sim 10:1:100$. These parameters describe pre-critical interaction regime.

The interaction point coincides with the center (x_c, y_c, z_c) of the domain $[0, 0.1] \times [0, 0.01] \times [0, 1.5]$. The focusing condition is described by the transformation between the laboratory system coordinates and the accelerator coordinate system:

$$x = X + (z - z_c)X', \quad y = Y + (z - z_c)Y', \quad z = Z + z_c,$$
$$p_x = p_z X', \quad p_y = p_z Y'.$$

In the numerical experiments, a usual grid is $100 \times 100 \times 100$, the number of macro-particles for such a grid should not be less than $J = 10^8$, the time step $\tau = 5 \cdot 10^{-5}$ is defined from the stability condition, and 14,000 time steps should be made.

The basic plasma physics equations. The mathematical model employed for the solution of the problem of beam relaxation in plasma consists of the Vlasov equations for ion and electron components of plasma and, also, of Maxwell's equation system. These equations in the usual notation have the following form:

$$\frac{\partial f_{i,e}}{\partial t} + \vec{v}\frac{\partial f_{i,e}}{\partial \vec{r}} + \vec{F}_{i,e}\frac{\partial f_{i,e}}{\partial \vec{p}} = 0, \quad \vec{F}_{i,e} = q_{i,e}\left(\vec{E} + \frac{1}{c}[\vec{v}, \vec{B}]\right),$$
$$\operatorname{rot}\vec{B} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}, \quad \operatorname{div}\vec{B} = 0,$$

$$\operatorname{rot}\vec{E} = -\frac{1}{c}\frac{\partial\vec{B}}{\partial t}, \quad \operatorname{div}\vec{E} = 4\pi\rho.$$

In the present paper, this equation system is solved by the method described in [3]. All the equations will further be given in the dimensionless form. The following basic quantities are used for the transition to the dimensionless form:

- characteristic velocity is the velocity of light $\tilde{v} = c = 3 \cdot 10^{10}$ cm/s;
- characteristic plasma density $\tilde{n} = 10^{14}$ cm^{-3};
- characteristic time $\tilde{t}$ is the plasma period (a value inverse to the electron plasma frequency) $\tilde{t} = \omega_p^{-1} = \left(\frac{4\pi n_0 e^2}{m_e}\right)^{-0.5} = 5.3 \cdot 10^{-12}$ s.

The Vlasov equations are solved by the PIC method. This method implies the solution of the equation of motion for model particles:

$$\frac{\partial\vec{p}_e}{\partial t} = -(\vec{E} + [\vec{v}_e, \vec{B}]), \quad \frac{\partial\vec{p}_i}{\partial t} = \kappa(\vec{E} + [\vec{v}_i, \vec{B}]), \quad \frac{\partial\vec{r}_{i,e}}{\partial t} = \vec{v}_{i,e},$$
$$\kappa = \frac{m_e}{m_i}, \quad \vec{p}_{i,e} = \gamma\vec{v}_{i,e}, \quad \gamma^{-1} = \sqrt{1 - v^2}.$$

The quantities with subscripts i and e are related to ions and electrons, respectively.

The leapfrog scheme is employed to solve these equations:

$$\frac{\vec{p}^{\,m+1/2}_{i,e} - \vec{p}^{\,m-1/2}_{i,e}}{\tau} = q_i\left(\vec{E}^m + \left[\frac{\vec{v}^{\,m+1/2}_{i,e} - \vec{v}^{\,m-1/2}_{i,e}}{2}, \vec{B}^m\right]\right),$$
$$\frac{\vec{r}^{\,m+1}_{i,e} - \vec{r}^{\,m}_{i,e}}{\tau} = \vec{v}^{\,m+1/2}_{i,e},$$

where τ is the timestep.

The scheme proposed by Langdon and Lasinski is used to obtain the values of electric and magnetic fields. The scheme employs the finite-difference form of the Faradey and the Ampere laws. A detailed description of the scheme can be found in [3]. The scheme is of the second order of approximation with respect to space and time.

3. Peculiarities of the ultrarelativistic dynamics simulation

The equations are highly nonlinear and require a good resolution. The beam-shape is also nonlinear in every direction: the particle distribution is Gaussian in the accelerator coordinate system. In addition, a beam significantly changes its shape in the coarse of time due to the focusing fields.

Thus, the majority of particles is concentrated in a small region at a certain moment.

Another problem in the computation of the 3D electromagnetic beam fields is a high value of the relativistic factors γ. Comparing with the case of non-relativistic motion the electric field of a relativistic particle γ times increases in the transversal direction and decreases γ^2 times in the longitudinal direction. The usual beam field computation for $\gamma = 10^3$ requires 10^9 operations and makes the computation prohibitive by standard methods even when using supercomputers. To overcome the above difficulties we consider the macro-particles as thin needles with the length h_z directed along the axis z [4]. The field contribution from the density, computed based on these pins, can be calculated as follows:

$$E_x(x,y,z) = \frac{2q(x-x_0)}{h_z((x-x_0)^2+(y-y_0)^2)}, \tag{1}$$

$$E_y(x,y,z) = \frac{2q(y-y_0)}{h_z((x-x_0)^2+(y-y_0)^2)}, \tag{2}$$

$$E_z = 0, \qquad \vec{H} = [\vec{v}, \vec{E}]. \tag{3}$$

4. Transition from the plasma physics code to the beam physics code

Let us start with the `GPUPlasma` class that implements the PIC method for GPU as applied to the relativistic plasma physics. The class has a special method for the field evaluation and particle pushing and initial distribution setting. In order to use all these factors with beam simulation, a derived class `GPUBeam` is created, as shown in Figure 1. As can be seen from the picture, the constructor of the GPUBeam class does nothing on its own, it just

```
template <template <class Particle> class Cell >
class GPUBeam: public GPUPlasma<Cell>
{

public:
    GPUBeam(){}
    ~GPUBeam(){}

    GPUBeam(int nx,int ny,int nz,double lx,double ly,double lz,
            double ni1,int n_per_cell1,double q_m,double TAU):
        GPUPlasma<Cell>(nx,ny,nz,lx,ly,lz,ni1,n_per_cell1,q_m,TAU){}

    void InitParticles(thrust::host_vector<Particle> & vp);

    void BoundaryConditions();
};
```

Figure 1. The screenshot of the GPUBeam class definition

```
int main(int argc,char*argv[])
{
   GPUBeam<GPUCell> *plasma;

   plasma = new GPUBeam<GPUCell>(100,100,100,1.5,0.01,0.1,1.0,100,1.0,5e-5);

   plasma->Initialize();

   plasma->Step();
```

Figure 2. The use of the GPUBeam class object

calls the constructor of the basic class. The parameters of the constructor represent the number of grid nodes, the domain size, the number of particles per cell, the average density, the charge-to-mass ratio and the time step.

The use of the defined class `GPUBeam` is quite simple (Figure 2). The `Initialize` function is called, which actually does all memory allocations and initializations, and then `Step` performs the time step. All the differences between the plasma simulation and beam interaction simulation are automatically taken into consideration through the mechanism of virtual functions.

Initial distribution. Since the initial distribution for the beam-beam interaction problem is entirely different, it is Gaussian instead of uniform, and, which is even more important, the particles occupy just a part of simulation domain, a new function should be constructed to implement the new distribution. In fact, the `InitParticles` function is made virtual in the basic class and is called from the `Initialize` method there. In the derived class, the `InitParticles` is properly redefined.

Boundary conditions are evaluated according to formulas (1), (2). In the plasma physics problem which was the test suite for the `GPUPlasma` class, the boundary conditions are periodic, thus a very simple CUDA kernel is called for just a few microseconds. Nevertheless, there are special wrapper functions that invoke this kernel. In the GPUBeam class, it is replaced by a more complex still very fast kernel, invoked by the `BoundaryConditions` function. The `BoundaryConditions` function is made virtual in the basic class and is called from the field evaluation method there. In the derived class, the `BoundaryConditions` function is properly redefined.

5. Performance

In the table, single time-step performance of the template implementation of the PIC algorithm for the beam-beam interaction (GPU-Beam) on a single Nvidia Tesla GPU is compared with CPU implementation (non-GPU) on Intel Xeon cores. The run here involves 1 million model particles and a grid of 100^3 nodes.

Computation time of a single time-step, ms

Operation	GPU-Beam	Non-GPU
Particle pushing	429.193	552.253
Field evaluation	536.250	180.426
Boundary condition computing	5.43	2893.723

As one can see from the table, boundary conditions computing time for the CPU version of the program is much greater. This is because of a complex set of send/receive operations for the CPU version of the program. And for the GPU version, the whole grid is situated in the memory of one single Tesla GPU and it does not need sending operations.

6. The GPU implementation

The implementation of the above PIC algorithm for the GPUs is quite standard. The field evaluation method is ported to GPU almost without any change. The computation speed is high enough even without optimization. The field arrays are stored in the GPU global memory.

The bottleneck of the PIC codes is a particle push. With the CPUs it takes up to 90 % of the runtime. So, first particles are distributed among cells. This step only two times reduces the push time with CPUs. With GPUs, it is even more important since it enables the use of the texture memory (the texture memory is limited and the whole particle array will never fit). The second step is keeping the field values related to the cell (and also to the adjacent cells) in a cell as it is. This is important since each particle needs 6 field values and writes 12 current values to the grid nodes, and now all this is done within a small amount of memory (a cell) without addressing the global field or current arrays that contain the whole domain. Then the evaluated currents from all the cells are added to the global current array.

This gives the speedup of about 10 for Tesla 2070 as compared to a single Xeon core. This is not so much, but no sophisticated optimization has been applied yet.

7. The template implementation of PIC method

In order to fulfil the main objective (a tool for the fast development of the new problem-oriented PIC codes for GPUs) it is necessary to do the following:

- develop an optimized GPU implementation of the PIC method for a particular problem,

- create a set of diagnostics tools to facilitate the analysis of results by the physicist, and
- provide an option to replace the problem-specific parts of a computational algorithm.

In order to do the latter, C++ templates are being used. This means that the "computation domain" class is implemented that contains "cell" class objects. The "cell" class contains "particle" class objects. Here "computation domain" is a template class with "cell" class as a parameter. The "cell" is a template itself, with "particle" as a parameter.

For a wide variety of PIC method implementations most of the operations of a cell with its particles are absolutely the same (adding/removing a particle, particle push, evaluation of a particle contribution to the current). Only a gridless particles method of gyrokinetic codes might be an exception. And even such codes fit the proposed scheme since they just do not need any operations, and do not introduce anything new. This fact gives a hope that these operations once implemented as a template will be efficient for a number of problems to be solved with the PIC method.

Also, operations of the computation domain with cells are absolutely the same. The things that differ are: the initial distribution and boundary conditions. Thus, these operations should be implemented as virtual functions.

Since particle attributes and operations with particles are similar in most cases, it is possible to create a basic implementation of the "particle" class containing the particle position, momentum, charge, and mass. Then, if for some new physical problem the "particle" needs new attributes, a derived class is implemented, and this new "derived particle" class is used as a parameter to the "cell" template class.

At present, there are object-oriented implementations of the PIC method (e.g. the OOPIC library), and also the template libraries for the PIC method [5, 6], however this is valid for the CPU-based supercomputers, but not for hybrid ones.

The porting of the implemented PIC method template to the GPU was done in the following way—on the basis of the PIC method template (Plasma class) a derived class was created (GPUPlasma class), which is also a template. In the GPUPlasma, the following methods were added:

- copying the domain to GPU,
- comparison of CPU and GPU results, and
- invocation of GPU kernels for field evaluation.

The implementation of the "cell" for GPU (GPUCell class) was inherited from the Cell class. It is important that particle storage within a cell must be

optimized in terms of the GPU memory, thus the structure or class arrays are not suitable. The GPUCell class also includes copying to and from a device and the comparison of GPU and CPU cells. The "particle" implementation for GPU here is exactly the same as for CPU. It is necessary to mention that for debugging the computation method should be implemented just once both for the CPU and the GPU.

References

[1] Snytnikov A.V., Romanenko A.A. Parallel template implementation of Particle-In-Cell method for hybrid supercomputers. // Bull. Novosibirsk Comp. Center. Ser. Computer Science. — Novosibirsk, 2014. — Iss. 36. — P. 79–89.

[2] Boronina M.A., Vshivkov V.A., Levichev E.B., at al. An algorithm for the three-dimensional modeling of ultrarelativistic beams // Numerical Methods and Programming. — 2007. — Vol. 8, No. 2. — P. 203–210.

[3] Vshivkov V.A., Grigoryev Yu.N., Fedoruk M.P. Numerical "Particle-in-Cell" Methods. Theory and Applications. — Utrecht-Boston: VSP, 2002.

[4] Vshivkov V.A., Boronina M.A. Three-dimensional simulation of ultrarelativistic charged beams dynamics: study of initial and boundary conditions // Math. Mod. — 2012. — Vol. 24, No. 2. — P. 67–83.

[5] Decyk V.K. Sceleton PIC codes for parallel computers // Comp. Phys. Comm. — May, 1995. — Vol. 87, Iss. 1–2. — P. 87–94.

[6] Malyshkin V.E., Tsigulin A.A. ParaGen — the generator of parallel programs for numerical models // Avtometriya. — 2003. — No. 3. — P. 124–135 (In Russian).