Bull. Nov. Comp. Center, Num. Anal., 17 (2015), 53–59

Particle reordering optimization in the Particle-in-Cell method implementation for the GPU-based supercomputers*

A.V. Snytnikov

Abstract. The main point of the performance of a code for a GPU (Graphical Processing Unit) is data locality. For the PIC method this means that all the particles belonging to one cell must be located closely in memory. During the particle push the particles might move to other cells, and must be transported to a different place in memory (to a different cell). This is called particle reordering.

For the first time the particle reordering technique is proposed that involves no critical sections, semaphores, mutexes, atomic operations, etc. This results in almost 10 times redution of the reordering time compared to the straightforward reordering algorithm.

1. Introduction

The objective of the present paper is to analyze and to increase the performance of the template implementation of the PIC method for the GPU, proposed in [1].

This work has been inspired by the effect of anomalous heat conductivity observed at the GOL-3 facility in the Budker Institute of Nuclear Physics [2]. The GOL-3 facility is a long open trap where the dense plasma is heated up in a strong magnetic field when injecting a powerful relativistic electron beam of a microsecond duration. The effect is a decrease in the plasma electron heat conductivity by 100 or 1000 times as compared to the classical value for the plasma with the temperature and density observed in the experiment. An anomalous heat conductivity arises because of the turbulence that is caused by the relaxation of a relativistic electron beam in the high-temperature Maxwellian plasma. The physical problem is to define the origin and mechanism of the heat conductivity decrease. This is of great importance for the fusion devices because the effect of anomalous heat conductivity helps in heating plasma and, also, in confining it. The problem of heat transport in fusion devices was widely discussed (e.g. [3, 4]) and some recent works [5].

It is also necessary to provide a large number of particles for each cell of the grid for the simulation of turbulence. The level of non-physical statistical fluctuations is inversely proportional to the number of particles per

*Supported by the RFBR under Grants 14-07-00241, 15-31-20150, and 14-01-31088 and, also, by the SB RAS Integration Projects 103 and 113.

cell. So, if there are too few particles, all the physical plasma waves and oscillations will be suppressed by the non-physical noise. Thus, the performance improvement will always be necessary, and this is the objective of this paper.

2. The model description

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 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 [6]. All the equations will be further 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:

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

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 Ampere laws. A detailed description of the scheme can be found in [6]. The scheme gives the second order of approximation with respect to space and time.

3. The GPU implementation

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

4. The template implementation of the PIC method

In order to accomplish 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 problem-specific parts of the computational algorithm.

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

For a wide variety of the PIC method implementations most of the operations of a cell with its particles are absolutely the same (adding/removing a particle, particle push, the evaluation of the particle contribution to the current). Only the gridless particles method or gyrokinetic codes might be an exception. And even they fit the proposed scheme since they just don't need some of the operations, they 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 solved with the PIC method.

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

Since particle attributes and the operations with particle, are similar in most cases, it is possible to create a basic implementation of the "particle" class containing particle position, momentum, charge, and mass. Then, if for some new physical problem the "particle" needs new attributes, then a derived class is implemented, and this new class "derived particle" 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 [7, 8], but for CPU-based supercomputers, 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 (class Plasma) a derived class was created (class GPUPlasma), 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 the GPU (GPUCell class) was inherited from the Cell class. It is important that the particle storage within a cell must be optimized in terms of the GPU memory, thus the structure

or the class arrays are not suitable. The GPUCell class also includes copying to and from a device and comparison of the GPU and the CPU cells. The "particle" implementation for the GPU here is exactly the same as for the CPU. Here it is necessary for debug that the computation methods are implemented just once both for the CPU and the GPU.

5. The bottleneck of the GPU implementation of the PIC method

Most of the time is usually spent on the particle pusher. But as can be seen from Table 1, in this implementation of the PIC method the particle reordering stage apperared to be the most time-consuming. The reason is the following. When a cell transmits its particles to another cell (writes the data to the particle array of this another cell), in the writing process one must be sure that nobody else is writing there at the same time. And this can easily happen since all the cells transmit their particles simultaneously. In such a way, a certain writing protection tool should be used. In the first implementation of the algorithm, a semaphore was given to each cell, and when it is "on", writing is prohibited. Thus, many cells are at the same time waiting to write their particles in some other cells which appear to be busy. And everybody waits...

6. Optimization of separate kernels

Particle pushing was previously used only in the global memory. But the profiling shows that this is a weak point. Due to this reason, the shared memory is used to save electromagnetic fields. There is no performance increase in pushing possibly because of some bigger problems, such as memory access patterns. This requires further analysis.

Particle reordering. The following scheme is proposed to reduce the reordering time:

1. The reordering kernel is divided into two parts;
2. Every cell constructs a list of particles that fly to other cells;
3. A 3D matrix of 3^3 size is filled which indicates to how many particles are going to fly to each neighbour cell (there are 26). Also, a position in the list must be provided;
4. A synchronization point, since all the cells should terminate;
5. Every cell calls its neighbour, learns how many particles are going to fly from the neighbour cells and where they are exactly located, and writes these particles into its own particle array.

Table 1. The result of optimization for the main stages of the GPU implementation of the PIC method, timing for Tesla M2090, ms

Stage	Not optimized	Optimized
Particle push	164.431	211.082
Field evaluation	0.272	1.195
Particle reordering	2049	220.623
Current assignment	19	22.67
Field assignment	90.4	11.983

One should notice that nowhere in this algorithm a need of using the same data at the same time arises. This means that all cells can work concurrently. That is why in the optimized version the reordering is much better.

The current and field assignment. This was previously sequentially done for each cell (each cell uses one CUDA thread). Using 125 threads (the size of a local field array in each cell) resulted in much faster work (the last line in Table 1).

7. Performance with different GPUs

In Table 2, the main parameters of the GPUs used in the optimization tests are shown. In Table 3, the time instant for different stages of the algorithm are given. The point of interest here is not just looking at these numbers, but analyzing difference and proposing a new optimization strategy. First, one can see that the field evaluation and assignment are faster with K40. This means that regular patterns are to be provided everywhere to attain good performance. Second, a push is performed almost 10 times slower with GeForce than that having even more cores than Tesla 2090. This means that a program mainly depends on the bus width and on the number of multiprocessors. Third, the current assignment time is close to Tesla and Kepler (Tesla K40). This means that not all the cores are used (Kepler has 5 times more!).

Table 2

Parameter	GeForce GTX 850M	Tesla M2090	Tesla K40m
Total amount of global memory, Mb	4096	5375	11520
Number of multiprocessors	5	16	15
Number of CUDA cores	640	512	2880
GPU max clock rate, MHz	902	1301	745
Memory bus width, bit	128	384	384

Table 3. Timing of the main stages of the algorithm with different GPUs (optimized version), ms

Stage	GeForce GTX 850M	Tesla M2090	Tesla K40m
Particle push	2000	211.082	285.839
Field evaluation	1.004	1.195	0.353
Particle reordering	2226	220.623	191.298
Current assignment	316	22.67	44
Field assignment	169	11.983	8.066

8. Conclusion

An optimization strategy for the GPU implementation of the PIC method is presented. A new particle reordering method is proposed with 10 times reduction of the reordering time.

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 Sci. — Novosibirsk, 2014. — Iss. 36. — P. 79–89.

[2] Astrelin V.T., Burdakov A.V., Postupaev V.V. Generation of ion-acoustic waves and suppresion of heat transport during plasma heating by an electron beam // Plasma Physics Reports. — Vol. 24, No. 5. — P. 414–425.

[3] Cohen B.I., Barnes D.C., Dawson J.M., et al. The numerical tokamak project: simulation of turbulent transport // Computer Physics Communications. — May, 1995. — Vol. 87, Iss. 1–2. — P. 1–15.

[4] Jaun A., Appert K., Vaclavik J., Villard L. Global waves in resistive and hot tokamak plasmas // Computer Physics Communications. — December, 1995. — Vol. 92, Iss. 2–3. — P. 153–187.

[5] Gardarein J.-L., Reichle R., Rigollet F., et al. Calculation of heat flux and evolution of equivalent thermal contact resistance of carbon deposits on Tore Supra neutralizer // Fusion Engineering and Design. — October, 2008. — Vol. 83, Iss. 5–6. — P. 759–765.

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

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

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