VESTNIK UDMURTSKOGO UNIVERSITETA. MATEMATIKA. MEKHANIKA. KOMP'YUTERNYE NAUKI
COMPUTER SCIENCE 2018. Vol. 28. Issue 3

MSC2010: 68U20, 68N99

© ***A. A. Romanenko, A. V. Snytnikov***

THE PECULIARITIES OF THE PARALLEL IMPLEMENTATION OF PARTICLE-IN-CELL METHOD

Particle-In-Cell (PIC) method is widely used for plasma simulation and the GPUs appear to be the most efficient way to run this method. In this work we propose a technique that enables one to speed up one of the most time-consuming operations in the GPU implementation of the PIC method. The operation is particle reordering, or redistribution of particles between cells, which is performed after pushing. The reordering operation provides data locality which is the key performance issue of the PIC method. We propose to divide the reordering into two stages. First, gather the particles that are going to leave a particular cell into arrays, the number of arrays being equal to the number of neighbor cells (26 for 3D case). Second, each neighbor cell copies the particles from the necessary array to its own particle array. The second operation is done in 26 threads independently with no synchronization or waiting and involves no critical sections, semaphores, mutexes, atomic operations etc. It results in the more than 10 times reduction of the reordering time compared to the straightforward reordering algorithm.

Keywords: optimization, GPU, simulation, PIC.

DOI: 10.20537/vm180311

Introduction

From the physical point of view 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 [1]. The GOL-3 facility is a long open trap where the dense plasma is heated up in a strong magnetic field during the injection of the powerful relativistic electron beam of a microsecond duration. The effect is a 100- or 1000-fold decrease in the the plasma electron heat conductivity compared to the classical value for the plasma with the temperature and density observed in the experiment. Anomalous heat conductivity arises because of the turbulence that is caused by the relaxation of the 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 to heat the plasma and also to confine it.

Like other plasma physics problems involving turbulence or, in general, non-Maxwellian velocity distribution, this problem requires kinetic treatment. It means either a direct finite-difference solution of the Vlasov equation or Particle-In-Cell method. While PIC method is noisy and time-consuming, still it is in many cases the only method that is able to treat the problem with no nonphysical simplifications.

There are a lot of PIC packages, including those adopted for GPU, e.g., PIConGPU [2], ALaDyn [3], and also some works devoted to particular problems of GPU implementation of the PIC method, for example, [4] and [5], but since they are all problem-specific, we make an implementation on our own.

The PIC method consists of two stages: field evaluation and particle pushing. Computational experiments with general purpose computers (e.g., the Lomonosov supercomputer at the Moscow State University) show that most time is spent on particle pushing (from 60% to 90%). Since the total time is large: 2 weeks with 128 Intel Xeon cores for a 3D problem with just $100 \times 4 \times 4$ mesh nodes and 10000 particles for each cell [6], and further research requires meshes of much larger size, it requires optimization. It is clear that the optimization must be aimed mostly at the particle pushing stage.

The first optimization approach might be the GPU implementation of the PIC method. Such an implementation resulted in a 30-fold speedup for Nvidia Kepler K40 compared to 4 cores of Intel Xeon. But it appears that further optimization is possible in order to make the GPU implementation more efficient.

The key performance issue for the PIC method is the data locality. This means that the particle must be stored in the cell with respect to its coordinate, because it facilitates the use of cache memory, and gives a speedup of 200% for general-purpose computers compared to the PIC method implementation with particle stored with no respect to their coordinate [7]. But the problem is that in the course of simulation the particles change their coordinates, as shown in Figure 1.

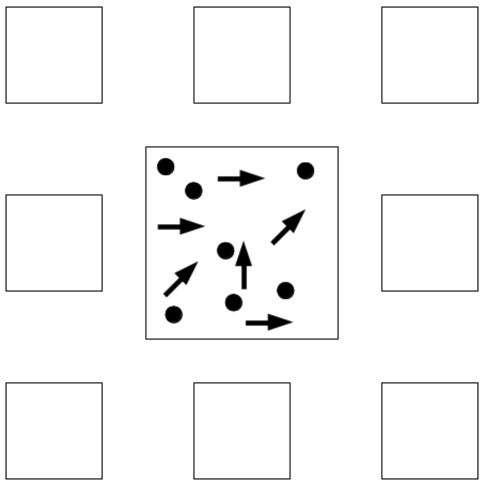

Figure 1. An example of particles in a cell after evaluating their new coordinates and impulses. The particles that stay in the cell are indicated by filled circles and the particles that must leave the cell are indicated by arrows pointing in the corresponding direction. Neighbor cells are shown with minor size squares. For simplicity, the 2D case is displayed.

Therefore, they must be transferred to other cells according to their new coordinates. In the present implementation this operation, called particle reordering, is performed involving atomic operations of CUDA.

Let us once more clarify the difference between particle pushing and particle reordering. Particle pushing is the evaluation of the new coordinates and impulses of the particles with no respect to the cell the particle belongs to. Particle reordering is performed after pushing and it virtually moves the particle (if necessary) to another cell in the cell array with respect to the particle's coordinate. Thus, pushing is moving the particle in the simulation domain and reordering is moving the particles in the cell array.

The atomic operations provide data integrity, but they make it impossible for more than one thread to work with particles of a cell. Moreover, it must be noted that reordering operation takes most of the PIC method simulation time for GPUs. In such a way we are going to describe an optimization which is capable of reducing reordering time.

This work continues the set of works on the optimization of PIC method implementation on GPU [8–11].

§ 1. Statement of the problem

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 the plasma and also of the system of Maxwell equations. In standard notation, these equation are

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

Here $f_{i,e}$ is the distribution function for ions and electrons, respectively, t is time, $\vec{v}$ is the velocity vector, $\vec{r}$ is the coordinate vector, $\vec{F}$ is the Lorentz force, $\vec{p}$ is the impulse, $\vec{E}$ is the electric field, $\vec{B}$ is the magnetic field, and ρ is the charge density.

Let us consider the following problem statement. The 3D computational domain has the shape of a parallelepiped with the following dimensions:

$$0 \le x \le L_X, \qquad 0 \le y \le L_Y, \qquad 0 \le z \le L_Z.$$

Within this domain there is the model plasma. The model plasma particles are distributed uniformly within the domain. The density of plasma and the electron temperature are set by the user. The temperature of ions is considered to be zero. Beam electrons are also uniformly distributed along the domain. Thus, the beam is considered to be already present in the plasma, and the effects that occur, while the beam is entering the plasma, are beyond the scope of this study.

The particles simulating beam electrons differ from those simulating plasma electrons by the value of their energy. Beam electrons initially have an energy of about 1 MeV, and the plasma electrons have an energy of about 1 keV. Moreover, beam electrons have one direction of movement strictly along the X axis, and plasma electron have the Maxwellian velocity distribution for all the three dimensions.

There is one more difference between the particles simulating beam electrons and plasma electrons. They have different weights when current and charge density is computed. Let us consider the ratio of beam density to plasma density, α (usually α varies from 10^{-3} to 10^{-6}), then the contribution of a beam electron particle is α from the contribution of a plasma electron particle. In such a way it is possible to provide a large number of beam particles.

§ 2. PIC method implementation for GPU

2.1. The idea of the PIC method.

Let us start with the problem written in the general operator form [12]:

$$\frac{\partial \mathbf{q}}{\partial t} + A\mathbf{q} = 0, \tag{1}$$

where q is some characteristic function of the system (for plasma dynamics or gas dynamics this is the distribution function), and A is the evolution operator.

The solution of the problem (1) is given by the following interpolation formula:

$$\mathbf{q} = \sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{Q_j R(u, u_j(t))}. \tag{2}$$

This is called the decomposition of the continuous medium into model particles. The function $R(\mathbf{u}, \tilde{\mathbf{v}})$ is called the particle form-factor. The interpolation formula (2) enables one to replace the solution of the problem (1) by the integration of the dynamic particle system.

2.2. Main principles of GPU implementation.

The programs for CPU and GPU are compiled from the source text. It is strongly necessary for debugging. The following main operations were implemented for GPU implementation:

- copying mesh arrays and particles from CPU to GPU;
- copying mesh arrays and particles from GPU to CPU;
- comparing GPU results to CPU simulations results when available.

The following operation remained on CPU:

- initial distribution of particles and fields;
- physical diagnostics and data output.

And the following operations are performed only by GPU:

- particle pushing;
- particle reordering;
- field evaluation.

Moreover, the present GPU implementation has the following differences from the common approach for PIC method:

- particles are stored in arrays attached to cells in order to use the cache more efficiently;
- field and current values (8 of them for each field and the current component) are stored in cells, that's why:
  - the current values are copied from each cell to the global 3D array used for field evaluation;
  - field values after evaluation are copied to each cell from the global field array.

2.3. Particle reordering operation.

The main time is usually taken by the particle pusher. But, in this implementation of the PIC method, the particle reordering stage appeared 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), the writer process 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, see Figure 2.

In such a way some write 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 . . .

§ 3. Optimization of particle reordering operation

The following scheme is proposed to reduce reordering time.

1. The reordering kernel (kernel is a function executed directly on GPU) is divided into two parts.
2. Every cell constructs a list of the particles that will fly to another cells.
3. Then a 3D matrix of 3^3 size is filled, which says, how many particles are going to fly to each neighbor cells (there are 26).
4. Then these flying away particles are copied to buffer arrays (a special buffer array for each of the 26 neighbor cells), see Figure 3.

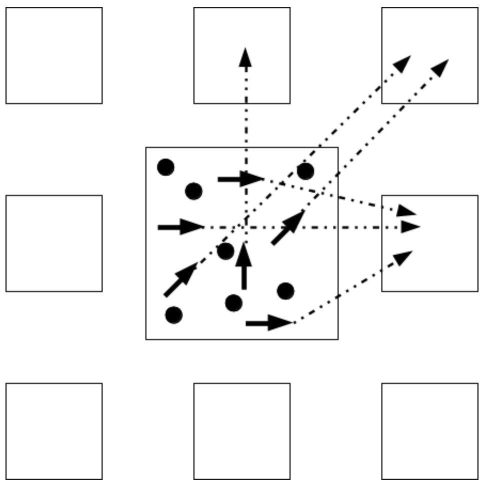

Figure 2. Read and write conflicts for particles that are being transmitted to neighbor cells. Each dash-dotted line demonstrates moving a particle to the neighbor cell. The particles that stay in the cell are indicated by filled circles and the particles that must leave the cell are indicated by arrows pointing in the corresponding direction. Neighbor cells are shown with minor size squares. For simplicity, the 2D case is displayed.

5. Synchronization point of time, since all the cells must finish this work.
6. Every cell calls its neighbor, gets how many particles are going to fly from that neighbor cells and where they are exactly located, and writes these particles to its own particle array.

One should notice that this algorithm does not need using the same data at the same time. This means that all cells can work concurrently. That's why in the optimized version reordering work much better.

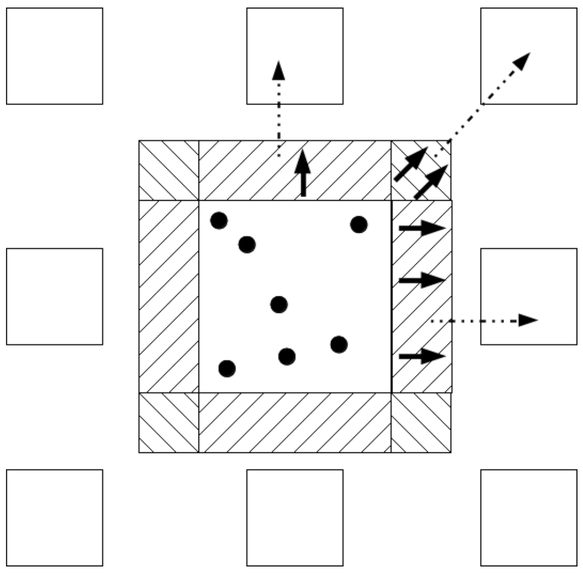

Figure 3. Read and write conflicts are resolved by means of buffer arrays (shown by hatching. Each dash-dotted line demonstrates moving a group of particles to the neighbor cell. The particles that stay in the cell are indicated by filled circles and the particles that must leave the cell are indicated by arrows pointing in the corresponding direction. Neighbor cells are shown with minor size squares. For simplicity, the 2D case is displayed.

§ 4. Results

In this section the result of the applied optimization technique is shown. It is shown by the example of Nvidia Kepler K40 GPU, for which we present both optimized version of the reordering algorithm (with no synchronization) and the old, non-optimized version (with synchronization). The time for the non-optimized version is shown by the bold font. For the modern GPUs (Kepler K80 and Pascal P100), there is only the optimized version. As it is seen from Table 1, the optimized version of the reordering operation works more than 10 times faster.

4.1. Dimensions and sizes.

The size of the 3D mesh in the presented computations is $N_x \times N_y \times N_z = 100 \times 4 \times 4$. In fact, 2 more nodes are added for each dimension to make the domain periodic, so it is $102 \times 6 \times 6 = 3672$ cells. The number of particles is 6.4 million for the whole domain, 1742 particles are average for each cell. The size of each 3D array in double precision is 30 Kilobyte, and since 9 such arrays are necessary (three components of electric field, magnetic field and current), it results in 0.3 MB. Each particle has 6 attributes, so the size of each particle is 48 bytes, and 6.4 million particles will need 300 MB of RAM. One can see that such test problem will fit into almost any GPU.

The size of all the buffer arrays per each cell is now set to 270 particles. In such a way it is 20% of the particles present, but one must note that this rather big buffer size must be set only for 3D problems involving turbulence, and is not necessary for steady flows.

The present 3D mesh size is just a test size. The real 3D plasma simulations will require at least 100^3 nodes with a similar number of particles per cell (~ 1000). Thus, in the real case, the size of the mesh array will be 48 MB, and for particles one will need 48 GB RAM, and considering the 20% overhead will need almost 50 GB. Thus, even for the modern Tesla V100 with 32 GB onboard one will need more than one GPU in order to conduct the real simulation. This is surely possible since MPI parallelization is implemented to the present code, yet not discussed in this paper.

Table 1. The main parts of the algorithm with different GPUs (optimized version) and also **non-optimized version** for Kepler K40 (the bold font); time is in microseconds

Device name	Kepler K40	Kepler K40	Kepler K80	Pascal P100
Reordering algorithm	**non-optimized**	optimized	optimized	optimized
Particle pushing	433.8	433.8	348.06	338.1
Field evaluation	31.7	31.7	25.4	19.3
Particle reordering	**27167**	1131.5	446.43	98.9

§ 5. Conclusion

An optimization technique for the GPU implementation of the Particle-in-Cell method is proposed. The technique eliminates synchronization from particle reordering operation. Synchronization is eliminated by splitting the reordering into two steps: first, construction of lists of the particles that will fly to another cells, and second, concurrently moving particles to neighbor cells. Since particle reordering appears to be the most time-consuming part of the particle pushing stage, and pushing takes up to 90% of the total simulation time, the proposed optimization results are in significant reduction of the total PIC computing time with GPUs.

Funding. This research was supported by the Russian Federal Targeted Program Grant 14.574.21.0145. Unique project ID – RFMEFI57417X0145.

REFERENCES

1. Astrelin V.T., Burdakov A.V., Postupaev V.V. Generation of ion-acoustic waves and suppression of heat transport during plasma heating by an electron beam, *Plasma Physics Reports*, 1998, vol. 24, issue 5, pp. 414–425.
2. Burau H., Widera R., Honig W., Juckeland G., Debus A., Kluge T., Schramm U., Cowan T.E., Sauerbrey R., Bussmann M. PIConGPU: a fully relativistic particle-in-cell code for a GPU cluster, *IEEE Transactions on Plasma Science*, 2010, vol. 38, issue 10, pp. 2831–2839. DOI: 10.1109/TPS.2010.2064310

3. Rossi F., Londrillo P., Sgattoni A., Sinigardi S., Turchetti G. Towards robust algorithms for current deposition and dynamic load-balancing in a GPU particle in cell code, *AIP Conference Proceedings*, 2012, vol. 1507, issue 1, pp. 184–192. DOI: 10.1063/1.4773692
4. Kong X., Huang M., Ren Ch., Decyk V. Particle-in-cell simulations with charge-conserving current deposition on graphic processing units, *Journal of Computational Physics*, 2011, vol. 230, issue 4, pp. 1676–1685. DOI: 10.1016/j.jcp.2010.11.032
5. Rieke M., Trost T., Grauer R. Coupled Vlasov and two-fluid codes on GPUs, *Journal of Computational Physics*, 2015, vol. 283, pp. 436–452. DOI: 10.1016/j.jcp.2014.12.016
6. Lotov K.V., Timofeev I.V., Mesyats E.A., Snytnikov A.V., Vshivkov V.A. Note on quantitatively correct simulations of the kinetic beam-plasma instability, *Physics of Plasmas*, 2015, vol. 22, issue 2, 024502. DOI: 10.1063/1.4907223
7. Tskhakaya D., Schneider R. Optimization of PIC codes by improved memory management, *Journal of Computational Physics*, 2007, vol. 225, issue 1, pp. 829–839. DOI: 10.1016/j.jcp.2007.01.002
8. Romanenko A.A., Snytnikov A.V., Timofeev I.V. 3D hybrid code for simulation of high-frequency electromagnetic radiation generation in turbulent plasma, *Vestn. Novosib. Gos. Univ. Ser. Inf. Tekhnol.*, 2016, vol. 14, issue 3, pp. 81–90 (in Russian). `https://elibrary.ru/item.asp?id=28098904`
9. Snytnikov A., Romanenko A. Parallel template implementation of Particle-in-Cell method for hybrid supercomputers, *Bulletin of the Novosibirsk Computing Center. Series: Computer Science*, 2014, vol. 36, pp. 79–88. `https://elibrary.ru/item.asp?id=25467821`
10. Snytnikov A., Romanenko A. The advantage of the GPU-based supercomputer simulation of plasma phenomena, *Bulletin of the Novosibirsk Computing Center. Series: Numerical Analysis*, 2013, vol. 16, pp. 81–92. `https://nccbulletin.ru/article/706`
11. Snytnikov A.V., Boronina M.A., Mesyats E.A., Romanenko A.A. A technology for the design of hybrid supercomputer simulation codes for relativistic particle electrodynamics, *Proceedings of the 1st Russian Conference on Supercomputing*, 2015, pp. 17–25. `http://ceur-ws.org/Vol-1482/017.pdf`
12. Grigoryev Yu.N., Vshivkov V.A., Fedoruk M.P. *Numerical "Particle-in-Cell" Methods*, VSP, Utrecht–Boston, 2002.

Received 23.05.2018

Romanenko Aleksei Anatol'evich, Candidate of Engineering, Associate Professor, Novosibirsk State University, ul. Pirogova, 1, Novosibirsk, 630090, Russia.
E-mail: arom@ccfit.nsu.ru

Snytnikov Aleksei Vladimirovich, Candidate of Physics and Mathematics, Researcher, Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of the Russian Academy of Sciences, pr. Acad. Lavrent'eva, 6, Novosibirsk, 630090, Russia.
E-mail: snytav@ssd.sscc.ru

А. А. Романенко, А. В. Снытников

Особенности параллельной реализации метода частиц в ячейках

Цитата: Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2018. Т. 28. Вып. 3. С. 419–426.

Ключевые слова: оптимизация, GPU, графические процессоры, моделирование, PIC.

УДК 519.684

DOI: 10.20537/vm180311

Метод частиц в ячейках широко используется для моделирования плазмы, в то время как графические процессоры представляются наиболее эффективным инструментом для проведения расчетов с помощью этого метода. В данной работе предлагается подход, позволяющий ускорить один из наиболее затратных по времени этапов в проведении расчетов по методу частиц в ячейках на графических ускорителях. Этот этап представляет собой переупорядочивание модельных частиц, или перераспределение их между ячейками сетки. Переупорядочивание модельных частиц позволяет обеспечить локальность данных, которая в первую очередь определяет эффективность реализации метода частиц в ячейках. В данной работе предлагается разделить переупорядочивание на два этапа. На первом этапе для каждой ячейки нужно собрать все модельные частицы, которые должны покинуть данную ячейку, в массивы, число которых равно количеству соседних ячеек (в трехмерном случае имеется 26 соседних ячеек). На втором этапе каждая из соседних ячеек копирует частицы из соответствующего массива рассматриваемой ячейки в ее собственный массив частиц. Так как второй этап может выполняться одновременно двадцатью шестью нитями без синхронизации и ожиданий, и при этом не используются критические секции, семафоры, мутексы, атомарные операции и другие подобные инструменты, то в результате время выполнения переупорядочивания сокращается более чем в 10 раз по сравнению с неоптимизированной реализацией переупорядочивания с использованием синхронизации.

Финансирование. Работа выполнена при поддержке Российской федеральной целевой программы (грант 14.574.21.0145, номер проекта — RFMEFI57417X0145).

Поступила в редакцию 23.05.2018

Романенко Алексей Анатольевич, к. т. н., доцент, кафедра систем информатики, Новосибирский государственный университет, 630090, Россия, г. Новосибирск, ул. Пирогова, 1.
E-mail: arom@ccfit.nsu.ru

Снытников Алексей Владимирович, к. ф.-м. н., научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН, 630090, Россия, г. Новосибирск, пр. Акад. Лаврентьева, 6.
E-mail: snytav@ssd.sscc.ru