Efficient Parallelization of the DSMC Method

P.V. Vashchenkov, A.V Kashkovsky and M.S. Ivanov

Institute of Theoretical and Applied Mechanics SB RAS, Institutskaya 4/1, 630090, Novosibirsk, Russia

Keywords: Domain decomposition, Parallel DSMC, Load balance algorithms
PACS: 07.05.Tp,05.10.-a,05.10.Ln

INTRODUCTION

The Direct Simulation Monte Carlo (DSMC) method is the most advanced tool for the numerical solution of rarefied gas dynamics [3]. This method models gas flows with the help of particles, which move and interact with each other and with the surface of the examined object like real gas molecules. As the number of molecules even in a strongly rarefied gas is very large, the same number of model particles cannot be used because of the limited computer performance. The number of model particles is always smaller than the number of molecules, but can still reach several million. Tracing the trajectories of millions of particles with allowance for mutual intersection is an extremely difficult challenge. It is often impossible to perform computations on a single-processor computer, and a parallel approach should be applied.

FUNDAMENTALS OF THE PARALLEL DSMC METHOD

The process of DSMC simulations is divided into two independent consecutive stages: transfer of all particles to a prescribed time step and collisions of particles with each other. Particles that can participate in a potential collision should be located in one cell of the computational domain. Collisions are performed on the basis of statistical information in a given cell. The true coordinates of the particles are no longer relevant.

Hence, the most convenient technique for parallelization of problems solved by the DSMC method is space decomposition. The computational domain is divided into zones in accordance with the number of processors, and each processor computes intermolecular collisions only in cells that belong to its subdomain. At the stage of particle transfer, the cell in which the particle is located after its motion is checked. If this cell belongs to another processor, the particle is removed from the current processor and moved to other processors.

The time diagram of one step of the parallel implementation of the DSMC method by an example of the SMILE++ software system is shown in Fig. 1. When the code is started, a copy of the ParalParent class is generated, which defines whether a single-processor or a multiprocessor version will be used in computations. In a multiprocessor version, it creates a MultiprocParalleler responsible for problem parallelization. For each processor, MultiprocParalleler creates buffers for sending particles (as particle transfer can occur from each processor to all other processors, the number of such buffers equals the number of processors) and one buffer for receiving particles.

Each processor generates particles in those cells (belonging to this processor) that are located at the input boundaries of the domain. To speed up code operation, the particles to be transferred to another processor are placed into the sending buffer SendBuffer.

Then there follows the Transmit() operation in which the sending buffers containing particles transmit them to the corresponding receiving buffers ReceiveBuffer. This procedure is finalized by the Sample operation where the particles received from the receiving buffer are transferred to the computational part of the code.

The collisional part of the computations is also performed locally on each processor. The last stage of the computational step of the SMILE++ parallel version is the stage of synchronization Synchronization.

If dynamic balancing is used, the loading of processors is analyzed after a certain number of computational steps. If the loading is insufficiently uniform, the computational domain is again divided into subdomains in accordance with the chosen algorithm.

The structure of classes that ensure this process can be seen in Fig. 2. In addition to the classes already considered, the figure also shows the AllocationAlgorithm class responsible for domain decomposition, the EffAnalizer class estimating the efficiency of parallel operation of the code, and the TimeCounter class.

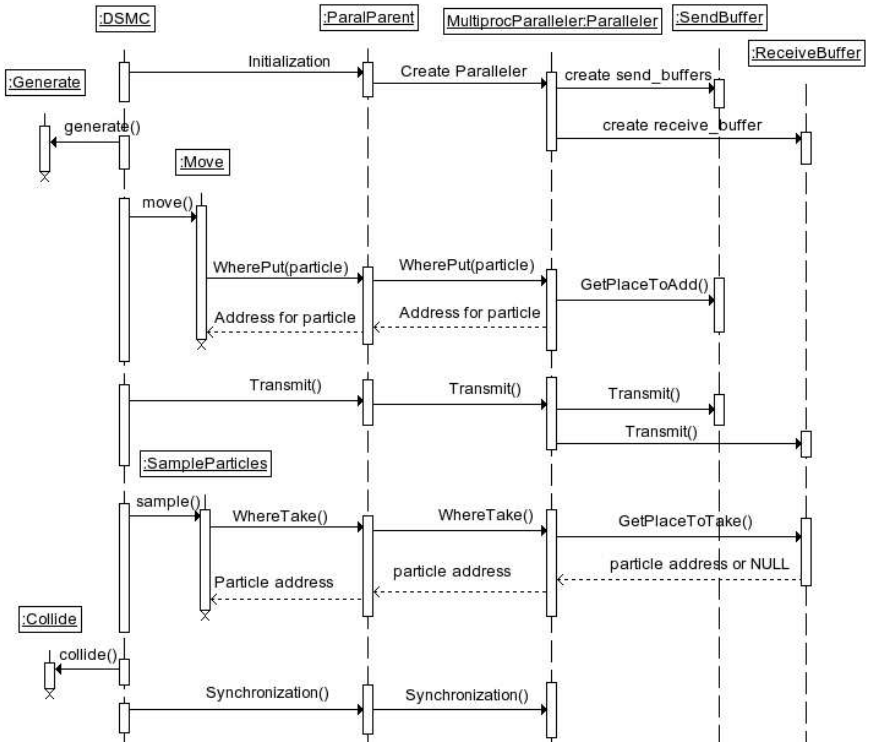

FIGURE 1. Time diagram of the parallel version of the SMILE++ software system

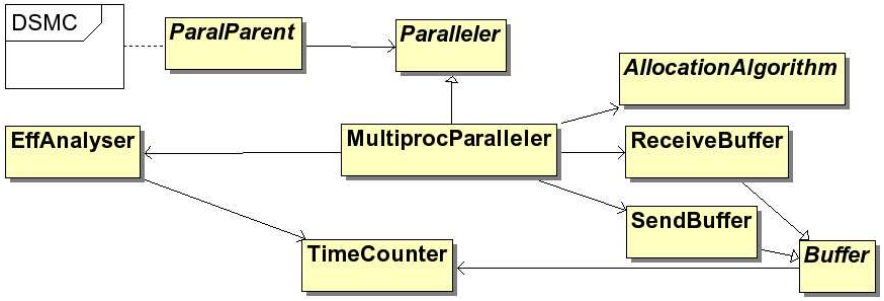

FIGURE 2. Parallelization diagram

Specific features of parallelization of problems of the DSMC method

Though the logic of parallelization of problems of the DSMC method is fairly simple, it is difficult to ensure a uniform balance of processor loading. The processor operation time needed for modeling intermolecular collisions depends on the number of particles in a cell. The density of cells in the computational domain, however, is not constant. It increases behind the shock wave, whereas there is a low-density region behind the body. Thus, it is a nontrivial problem to divide the computational domain into zones providing uniform loading of the processors in problems of the DSMC method.

Let us consider several algorithms of domain decomposition and their operation in the SMILE++ software system by an example of computing a two-dimensional gas flow around a cylinder (Knudsen number 0.01 and velocity $M = 5$). Figure 3 shows the density field around the cylinder. The values of density in different regions of space are seen to be strongly different.

The computations were performed on a MVS-1500 computational cluster of the Joined Supercomputer Center (Moscow, Russia). This cluster has 1070 PPC970FX processors and a Mirynet network (2 GBit/s).

Let us consider operation of all algorithms by an example of 24 processors.

PROBABILITY ALGORITHM

The simplest static algorithm providing very good balancing of processor loading is the probability algorithm. It is used in the SMILE software system [4].

The algorithm implies that the cells are randomly distributed between the processors. Thus, there is a large probability that each processor will have cells both from the dense flow and from the rarefied flow regions. Because of such a distribution of cells, all processors are uniformly loaded. The distribution of cells obtained by the probability algorithm of domain decomposition is illustrated in Fig. 4. This algorithm is simple in implementation and offers effective balancing of processor loading for all problems. Yet, its bottleneck is a large volume of data transfer between the processors. Indeed, in using this algorithm, there is a high probability that two neighboring cells belong to different processors. Thus, it is necessary to transfer almost all cells to other processors at each time step. For a cluster with a network interface between the processors, this fact may become a severe decelerating factor. For this reason, dynamic algorithms for domain decomposition were proposed.

FUNDAMENTALS OF DYNAMIC DECOMPOSITION OF THE COMPUTATIONAL DOMAIN IN PROBLEMS OF THE DSMC METHOD

The dynamic algorithms proposed imply the following rules:
1) the processor load depends on the number of particles belonging to this processors;
2) the volume of data transfer is determined by the flux of particles through the boundary between the responsibility zones of the processors.

Based on these rules, the algorithms of efficient domain decomposition should satisfy the following conditions:

1. All processors should have an identical number of particles;
2. The flux of particles through the boundaries between the responsibility zones of the processors should be as low as possible.

All algorithms proposed involve a preliminary stage with arbitrary decomposition of the computational domain. The objective of this stage is obtaining the density and velocity distributions in all cells.

Based on this information, each algorithm divides the computational domain in its own manner.

ALGORITHM OF DIVISION INTO IDENTICAL PARTS

The number of processors is divided into prime numbers. In our case, these prime numbers are $2 \times 2 \times 2 \times 3$. The domain is divided along one coordinate axis into two subdomains with an identical number of particles. The choice of the axis for this division is made under the condition of the minimum flux of particles through the boundary. Each subdomain is again divided into two parts with an approximately equal number of particles and the minimum flux of particles. Then each subdomain is again divided into two parts, and finally, each resultant subdomain is divided into three parts. Thus, we obtain the domain divided into 24 subdomains with an approximately identical load on each processors and a minimized volume of data transfer. The results of domain decomposition by the this method are illustrated in Fig. 5.

ALGORITHM ”LIFE“

The name of this algorithm originates from the famous computer game called ”Life“, which modeled reproduction of a colony of microorganisms. The algorithm operates as follows. A certain number of cells are chosen in the computational domain, and each cell is allocated to a particular processor. They form the responsibility zones of the processors. After that, the zone with the minimum load is found, and the next yet non-allocated cell with the maximum flux of particles through the boundary is attached to this zone. Thus, the responsibility zones of the processors gradually

TABLE 1. Results of computations on different numbers of processors of the MVS-1500 cluster

Algorithm	Computation time (s)					
	1 Proc.	2 Proc.	4 Proc.	8 Proc.	16 Proc.	32 Proc.
Probability		10152	9365	4992	2633	1075
Division		10856	6184	3711	2007	1545
Life	17564	9159	5906	3959	1737	1550
Moving boundaries(1)		9935	5914	3059	2444	1238
Moving boundaries(2)		10392	5982	3839	2444	1837

cover all free cells, tending to retain the maximum flux of particles between the cells inside the zone. The result of domain decomposition by this algorithm is shown in Fig. 6.

ALGORITHM ”MOVING BOUNDARIES“

This algorithm operates as follows. The density and velocity fields are known. The computational domain is divided into zones with an identical number of cells. Then the most severely loaded processor is chosen. Among its boundary cells, the cell with the maximum flux of particles is chosen and transferred to the processor to which the flux is directed.

Such an operation (search for the processor with the maximum load, search for the boundary cell with the maximum flux of particles, and transfer of this cell to another processor) is continued until the disbalance of processor loading reaches an admissible value.

Two options of the initial distribution were considered: along and across the flow. The results are plotted in Figs. 7 and 8.

COMPARISON OF THE EFFICIENCY OF THE ALGORITHMS

Let us consider the computation times needed to solve the problem of the flow around a cylinder on different numbers of processors with the use of different algorithms of domain decomposition. The results are summarized in Table 1.

It is seen from Table 1 that the efficiency of almost all dynamic algorithms, except for the algorithm ”Moving boundaries“ with the initial division of the domain across the flow, is higher than the efficiency of the probability algorithm. The unexpected advantage of the static probability algorithm over the dynamic algorithms in the computation on 32 processors is due to a too large number of steps chosen between the stages of adaptation of the computational domain. At the unsteady stage of the flow, dynamic algorithm did not have enough time to capture changes in the flow. efficiency of the probability algorithm.

CONCLUSIONS

Algorithms of decomposition of the computational domain for increasing the efficiency of operation of parallel codes for solving problems by the DSMC method are proposed. A significant decrease in computation time is demonstrated by an example of using 2, 4, 8, and 16 processors. Unexpected results obtained with the use of 32 processors indicate the necessity of a more detailed analysis of the algorithms of the DSMC method. Different values of algorithm efficiency obtained on different numbers of processors suggest that a tool can be created for using several dynamic algorithms with the most suitable one being chosen in solving particular large problems.

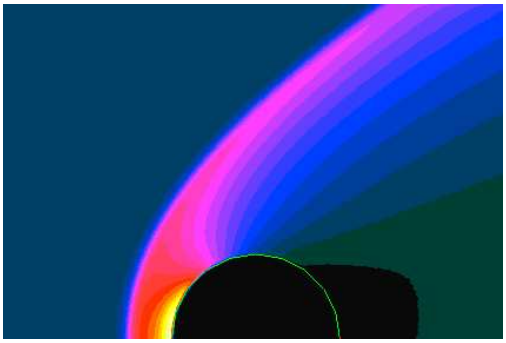

FIGURE 3. Density field around the cylinder. Kn = 0.01, M = 5.

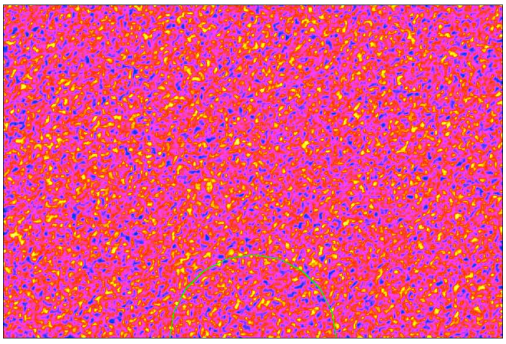

FIGURE 4. Result of probability algorithm work

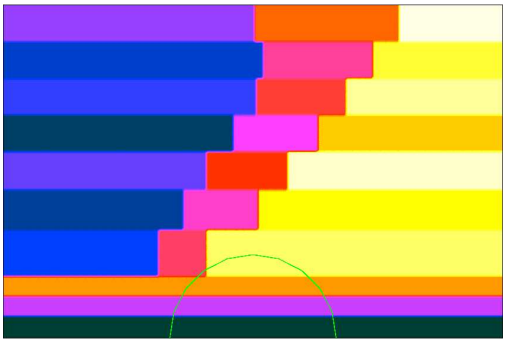

FIGURE 5. Result of algorithm with division into identical parts work

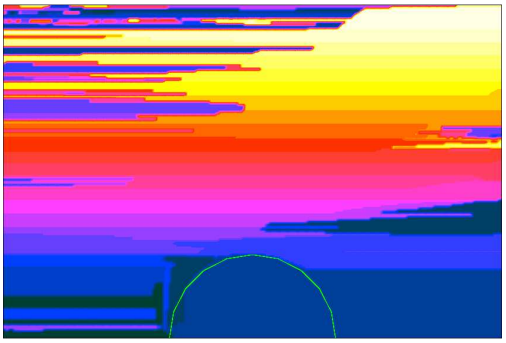

FIGURE 6. Result of "Life" algorithm work

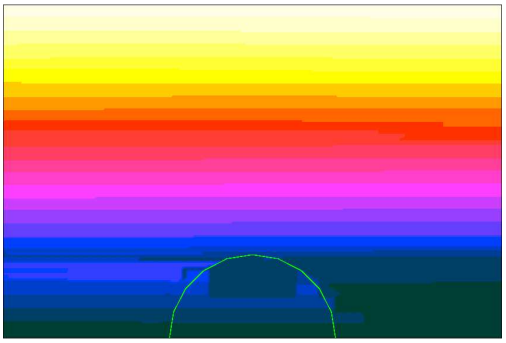

FIGURE 7. Result of "Moving boundaries" algorithm work. Primary division along the flow

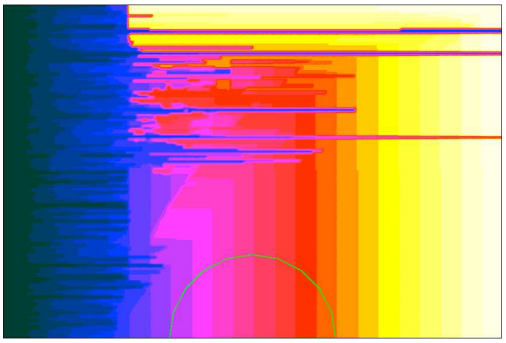

FIGURE 8. Result of "Moving boundaries"algorithm work. Primary division across the flow

REFERENCES

1. A.Kashkovsky, G.Markelov and M.Ivanov, An Object-Oriented Software Design for the Direct Simulation Monte Carlo Method, AIAA 2001-2895.
2. A.V. Kashkovsky, Ye.A. Bondar, G.A. Zhukova, M.S. Ivanov, S.F. Gimelshein, Object-Oriented Software Design of Real Gas Effects for the DSMC Method, Rarefied Gas Dynamics: 24th Int. Symp., Porto Giardino, Italy, July 10-16, 2004, (Ed. M.Capitelli), AIP Conference proceedings, Vol. 762, 2005, pp.583-588.
3. G.A. Bird. Molecular Gas Dynamics and the Direct Simulation of Gas Flows: Clarendon Press, Oxford, 1994.
4. Ivanov M.S. and Markelov G.N. and Gimelshein S.F. Statistical simulation of reactive rarefied flows: numerical approach and applications, AIAA Paper 98–2669, 1998