Parallel Object-Oriented Software System for DSMC Modeling of High-Altitude Aerothermodynamic Problems

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

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

Keywords: DSMC, Software design
PACS: 47.45-n 02.60.Cb 01.50.hv

INTRODUCTION

A wide range of problems in both science and industry, starting from micronozzles and extending to large aerodynamic configurations and complex fundamental investigations of rarefied flows, have been solved using the Direct Simulation Monte Carlo [1] (DSMC) method. Thus, the ever-growing need for a robust, reliable, easy-to-use, and well-validated code for DSMC computations is clearly seen. Such a code should be well-documented and suitable for users without specialized training in DSMC techniques.

There exist a number of both freeware and commercial packages that address these needs for numerically and ergonomically efficient DSMC systems. These codes include DS2V/3V [2] developed and maintained by G.A.Bird, MONACO [3] developed by Iain D. Boyd et al., and DAC [4] developed in Johnson Space Center. The development of a powerful software system based on the DSMC method was started at the Laboratory of Computational Aerodynamics (ITAM, Novosibirsk, Russia) at the end of the 1980s, being concentrated on the following subjects: mathematically grounded numerical schemes for the DSMC method [5]; collision models for description of real gas effects; parallel algorithms for the DSMC method and special techniques for decreasing the computational cost; pre- and post-processing systems that simplify the entire cycle of aerodynamic studies, from geometric model definition to visualization of results. This research resulted in the creation of the SMILE system (Statistical Modeling In Low-Density Environment) [6] capable of solving a very wide range of basic and applied problems of rarefied gas dynamics. The SMILE core code is written in FORTRAN90 and has no memory limitations specific to static FORTRAN programs.

Our experience during the development of the SMILE software system clearly demonstrates the fact that the greater the capabilities of the DSMC system are, the harder it is to modify the system by implementation of new models, methods and algorithms. The necessity for the creation of a DSMC software system based on the new-generation Object-oriented programming (OOP) approach became evident to us several years ago.

The main goal of the present paper is to describe the basic ideology and demonstrate example applications of the SMILE++ parallel software system [7] developed at ITAM. The SMILE++ is based on the OOP approach and is completely written in C++. It is the descendant of SMILE and incorporates most of the capabilities of the latter. At the same time, it has new capabilities and significant advantages offered by OOP.

OBJECT-ORIENTED APPROACH IN DSMC

Specific features of high-altitude aerothermodynamics of space vehicles impose specific requirements on software applications used in this field: the need for generating and using complicated three-dimensional geometric models and grids; possibility of using various collisional models of physicochemical processes and various gas/surface interaction models; availability of a database of chemical elements, their molecular properties and parameters determining their collisional interaction with other chemical elements; possibility of automation of the parallelization process while retaining high efficiency of parallelization; possibility of using a multizone approach and boundary conditions of various types. The necessity to satisfy all these requirements makes computational systems extremely cumbersome and sophisticated. On the other hand, the software system should be readily modified for new capabilities. Simultaneously,

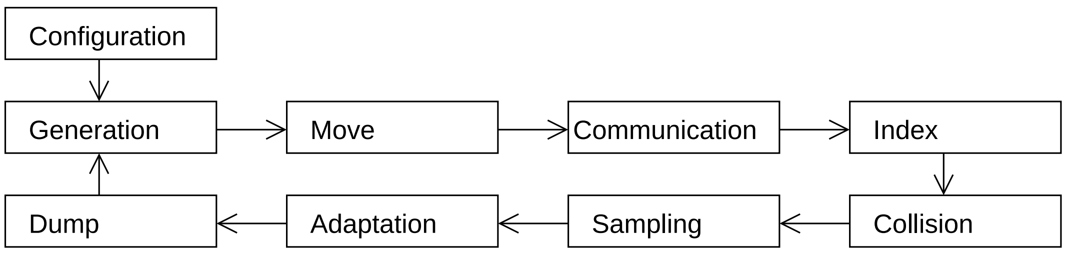

FIGURE 1. Diagram of DSMC computation.

it is important to retain all already available capabilities of the system. Object-oriented programming (OOP) is a key mechanism in this task.

OOP implies that the code is a model of a real process as a set of interacting objects. The object is described by a number of parameters whose values determine the state of the object and the set of operations (actions) that can be performed by the object. It is important that there can be several replaceable objects performing identical (or similar) operations, but by different methods. If each object is responsible for a certain model of physical phenomena (e.g., different types of chemical reactions), then one can configure the code from properly chosen objects (physical models) for a particular problem, like a bricklayer makes a wall from bricks.

As the objects are rather closed structural units of the code, it is more convenient to use OOP in large-scale codes developed by a team of programmers. As the interactions between the objects (actions) are limited (the number of actions usually does not exceed 20) and are specified at an early stage of code design, the code for each object is created independent of other codes. This approach substantially simplifies the code development and prevents erroneous usage of data stored for other purposes.

The most important problem in using OOP is optimal separation of examined phenomena into individual objects and definition of properties and actions of these objects. Provision of the minimum possible interactions between the objects facilitates replacing them by their inheritors and simplifies code modification.

The main objects in the DSMC++ code discussed are:

Species – properties of the gas species,
Mesh – geometric parameters of the mesh cell and methods of obtaining the cell number from specified coordinates,
Cell – mesh cell and its properties,
Particle – particle properties,
Geometry – description of the geometric shape and surface properties of the spacecraft under study.

All these objects can be presented as something material or existing. There are other objects, however, which are responsible for actions with other objects rather than displaying some material properties:

Generation – generation of new particles,
Move – motion of particles,
Collision – collision of particles,
Sampling – sampling of macroparameters,
Paralleler – interaction with other processors in multiprocessor computations.

Note that the mesh and the cells are different objects weakly related to each other. The reason is that the mesh cells usually contain statistical information not related to a particular shape of the cell. Therefore, the mesh shape and its dimension (2D/Axisymmetrical or 3D) may be changed, while the contents of the cells remain unchanged.

The overall diagram of computations (actions performed with objects) is shown in Fig. 1. The object *Configuration* reads initial data from a file; depending on these data, it generates all necessary objects in the computer memory and performs all preparatory computations. Each generated object can address the container of particles or cells and order properties needed for its further operation.

Each time step begins from generation of new particles by the object *Generation*. Depending on the problem to be solved, one of the inheritors of this object is used, which generates particles for the 2D, 3D, or axisymmetric case. In addition, particles can be generated on the domain boundary (external flow), on a specified surface inside the domain (e.g., the starting surface of the jet).

By means of the action *Move*, all particles are moved to a given time step. As previously noted, a particular inheritor is used for the 2D, 3D, or axisymmetric case. In the case of a particle collision with the spacecraft geometry, particle reflection is modeled, and momentum imparted to the surface is calculated. This momentum is summed up and is further used to obtain the forces and moments acting on the spacecraft. Reflection is modeled, for instance, by the Nocilla model or by the specular-diffuse model, each of them being represented by its own object generated during task configuring. In addition, the particle may fall into a cell that belongs to another processor. In this case, such a particle is transferred to the *Processor*, which stores particles in a buffer for subsequent transfer to other processors. Thus, any object *Move* interacts with the particles, mesh, geometry, and processor.

After all particle are moved, the action *Communication* is performed, which transfers the particles from the buffer through the *Processor* to other processors and receives the particles from other processors. If the computations are performed on one computer, then a single-processor inheritor *Processor* is used, which only emulates all operations. Then, the action *Index* is performed, which determines which particles are located in each particular cell. This object has inheritors that perform indexation for a one-species gas and for a mixture of gases with different methods of storing the indices.

The action *Collision* performs collisions of particles in the cell. This action exists in the one-species and multi-species variants. The single-species version involves, for instance, the VHS or VSS model of modeling post-collision velocities and models for exchange of internal energy with continuous or discrete energy. The multi-species variant has additional objects responsible for chemical reactions [8]. All of them are inherited from a certain basic object *Reaction* and can perform dissociation, recombination, or exchange reactions.

The action *Sampling* accumulates statistical information about the particles in the mesh cell. This information is used as a basis for calculating gas-dynamic parameters of the flowfield. Based on the accumulated statistical information, the computational mesh is periodically adapted to the flowfield. Adaptation means constructing a second-level mesh in the cell. As the geometric size of the initial mesh remains unchanged, the size of second-level cells is uniquely determined by the level of splitting. The second level is needed only for more correct modeling of particle collisions; therefore, it is used only in the action *Collision*. All other objects are not related to the second level. The current state of the system is periodically written into files (action *Dump*). These files are used to continue the computations if they were interrupted for some reason and to obtain flowfields, which is done by a separate program.

The OOP advantages are not visible for the final user. For developers, however, it is much simpler to add new physical models. For instance, the class *Container* is used to store particle properties, and the list of properties can be different, depending on the problem to be solved. The properties of each particle are located in a continuous segment of memory. Arrangement of particle properties in this fragment, as well as storage, extraction, and transformation of types, is ensured by the container. The number of particles can change in the course of computations, but the set of properties within one problem is identical for all particles. The structure of properties being stored is created before the beginning of computations. During problem configuring, each physical model addresses the container and orders a set of particle properties necessary for operation of this model (see Table 1).

The container remembers the type and length of each ordered property, the relative address from the particle beginning, and the key identifying this property (see Fig. 2). The sum of all lengths of the properties determines the total length of the particle. During the computations, the physical model can extract and modify the corresponding data for the required particle using this key. The particle is transferred to another processor by copying the memory fragment occupied by this particle. The container also provides storage of properties in a file and loading of properties from this file when the computation is restored.

TABLE 1. Particle properties required by different physical models.

Property	Move		Collision		Internal Energy	
	2D/Axisym.	3D	Single	Mixture	Continuous	Discrete
V_x,V_y,V_z	×	×	×	×		
X,Y	×	×				
Z		×				
Component				×		
E_{rot},E_{vib}					×	
L_{rot},L_{vib}						×

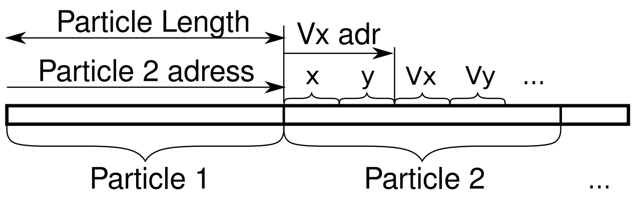

FIGURE 2. Structure of storage of particle properties.

A new physical model that requires additional particle properties is added in the following sequence:

1. a file containing the description of the class of this model and the methods (functions) of this class is created;
2. the initialization function is supplemented with a response to key words in the initial data file, which creates an object of this class and invokes its initialization function. Necessary particle properties should be ordered in this function;
3. initialization of the corresponding method is added in the function of collisions (moving, or any other place where this physical model is used).

In this sequence, all changes are concentrated in two or three places of the code. If the particle properties were stored in a usual data array, then addition of a new property would change the array dimension, which would have to be traced in all functions where this array is used. This would be a source of possible program errors, which can be avoided by using OOP.

PARALLELIZATION

An important specific feature of the DSMC method is a rapid growth of its computational cost with increasing free-stream density. Moreover, it is possible to state now that the possibilities for further improvement of numerical efficiency of the DSMC method by modification of its numerical schemes have been exhausted. Therefore, the only realistic way to increase the efficiency of the method for aerothermodynamic applications is its parallelization. In particular, it should be noted that modeling of real three-dimensional flows around space vehicles at flight altitudes of about 80 km (in the near-continuum regime) is impossible without using multiprocessor computers.

The SMILE++ code parallelization is based on the domain decomposition concept where each processor operates only with some part of the cells of the computational domain and with particles located in these cells. If a particle after moving falls into a cell that belongs to another processor, the particle should be transferred to this processor. Domain decomposition can involve a dynamic algorithm, which periodically re-distributes the cells between the processors, based on accumulated statistical information. In this case, all information that refers to a particular cell is transferred to a new processor.

The time diagram of parallel implementation from the DSMC method by an example of the SMILE++ software system is shown in Fig. 3. The computational code for parallelization invokes functions of an object of the class ParalParent. A copy of the class MultiprocParalleler is created inside this class in the case of a multiprocessor problem, and a copy of the class SingleParalleler is created for a single-processor problem. SingleParalleler emulates a situation where all particles and cells belong to one processor, and no particle or cell re-distribution is needed. MultiprocParalleler creates buffers for particle sending SendBuffer for all other processors and one buffer for particle receiving ReciveBuffer. A processor map is also generated with indications of which particle belongs to which processor.

Each time step includes generation of new particles and moving of all particles on a processor. A cell in which the particle is located after its moving is determined for each particle. If this particle belongs to another processor, its address in the corresponding buffer is requested from ParalParent; the particle is copied to this address and is deleted from the previous processor. When the buffer is filled, the particles are transferred by the operation Transmit(). After moving all particles and transmission of all buffers, there follows the operation Sample, in which particles are received from all processors to the receiving buffer and then transferred to the computational part of the program. When all particles are obtained, intermolecular collisions are performed, statistical information is accumulated, etc. At the end of the time step, the computations are synchronized: the system waits for a state where all processors finish this

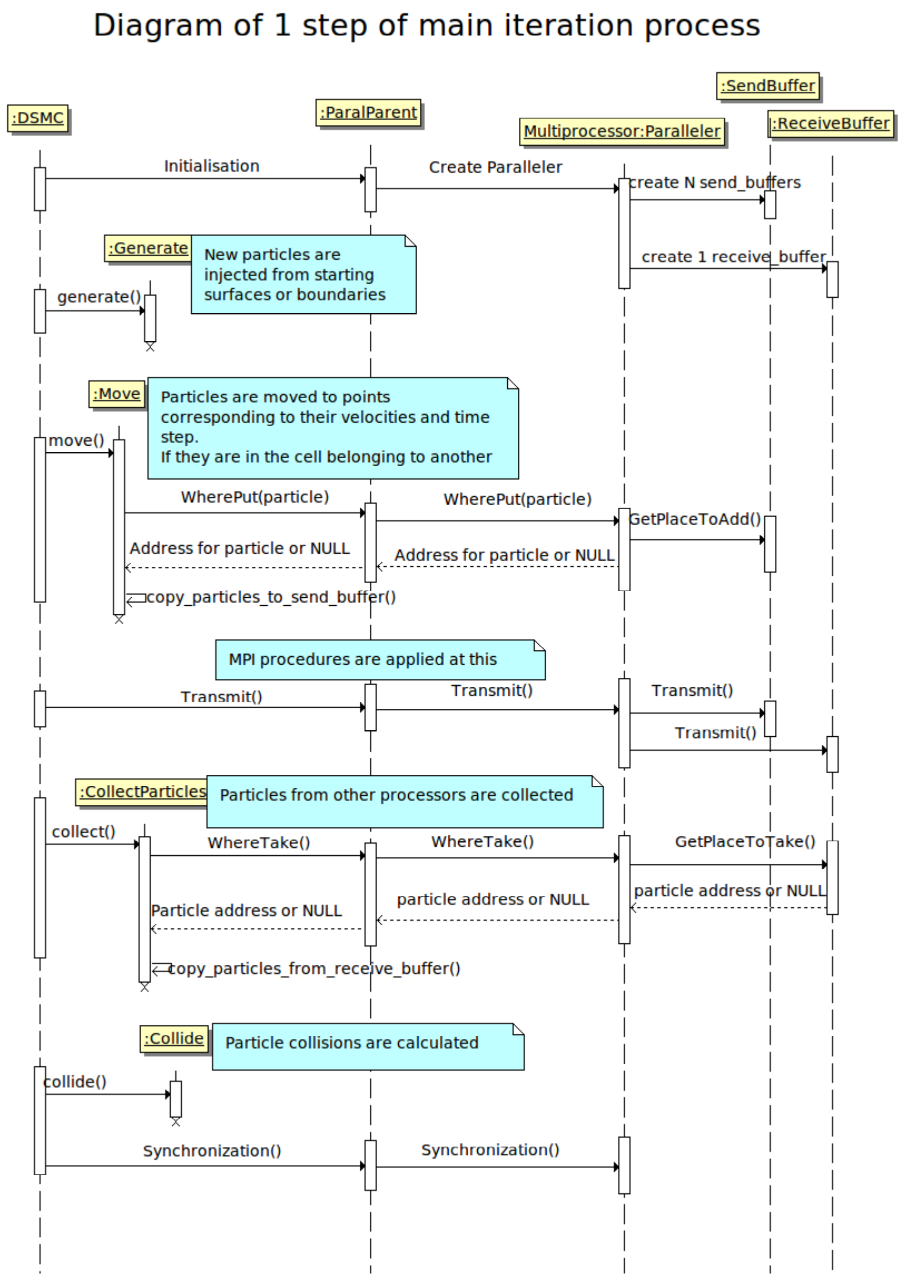
Diagram of 1 step of main iteration process
:DSMC
:ParalParent
Multiprocessor:Paralleler
:SendBuffer
:ReceiveBuffer
Initialisation
Create Paralleler
create N send_buffers
create 1 receive_buffer
:Generate
New particles are injected from starting surfaces or boundaries
generate()
:Move
Particles are moved to points corresponding to their velocities and time step. If they are in the cell belonging to another
move()
WherePut(particle)
WherePut(particle)
GetPlaceToAdd()
Address for particle or NULL
Address for particle or NULL
copy_particles_to_send_buffer()
MPI procedures are applied at this
Transmit()
Transmit()
Transmit()
Transmit()
:CollectParticles
Particles from other processors are collected
collect()
WhereTake()
WhereTake()
GetPlaceToTake()
Particle address or NULL
particle address or NULL
particle address or NULL
copy_particles_from_receive_buffer()
:Collide
Particle collisions are calculated
collide()
Synchronization()
Synchronization()

FIGURE 3. Time diagram of particle exchange.

step. Without synchronization, it may happen that a processor that has few particles performs all motions and transfers particles to a processor with a greater load, which is still busy with the previous step.

If dynamic balancing of processor loading is used, then the load distribution is analyzed after a certain specified number of steps. If the loading is not uniform enough, then the domain is again divided into subdomains in accordance with the chosen algorithm. After that, similar to particle transfer, the cells are transmitted to the corresponding processors, and the processor maps are corrected. The SMILE++ system employs various algorithms of dynamic load balancing, which distribute the load in identical portions between the processors and minimize the time of exchange between the processors. They include the "life" algorithm, the algorithm of division into identical parts, etc. Depending on the problem to be solved and on the number of processors used, different algorithms are more efficient.

GRAPHIC USER INTERFACE

In addition to the computational code, a large amount of initial information has to be prepared for computations: description of gas properties, geometric model, free-stream parameters, parameters of methods used, etc. After computations, results have to be processed and presented in a form convenient for analysis. For this purpose, the Graphic User Interface (GUI) was developed. The GUI is a superstructure over a set of programs, which can be divided into the Pre-processing subsystem (programs for preparing data), Post-processing subsystem (program for processing results), and Processing subsystem (computational programs and programs for monitoring the computational process).

The Pre-processing subsystem includes:

- a database of chemical elements, molecular collisions, and chemical reactions;
- a subsystem for geometric modeling designed for creating the model of the geometric shape of the spacecraft surface and defining its physical properties. As aerodynamic computations require only the shape of the spacecraft surface, its inner structure is not generated. Geometric modeling of complex-shaped spacecraft surfaces is based on the principle of the element-by-element description. Each spacecraft is presented as a set of basic elementary surfaces called primitives. The primitives can be flat elements (rectangle, circle, etc.), fragments of surfaces described by second-order equations (cone, sphere, paraboloid, etc.), and surfaces defined by sections or by a set of triangles. There is a particular triangulation program for each primitive.
- an Adviser program, which gives recommended values of the DSMC method parameters.

The Processing subsystem includes:

- a DSMC computational module;
- utilities that control the computation process, monitor the convergence of the solution, etc.
- utilities for changing the number of processors, restarting the statistical sampling, and changing the parameters of the numerical method.

The Post-processing subsystem includes:

- utilities for processing of dump files and obtaining flowfield parameters (in Tecplot and vtk formats), surface distributions, distribution functions etc.;
- built-in tools for flowfield visualization;
- utilities for converting computed results into initial data for another computation.

EXAMPLES OF SMILE++ APPLICATION

Clipper spacecraft

A specific feature of the ”Clipper“ spacecraft is its configuration: it has a lifting body shape, and its wings have tip fins to increase the lift force and control spacecraft motion in terms of the yawing angle. Computations were performed to determine the aerothermodynamic characteristics of this spacecraft at altitudes from free-molecular flight to 95 km. As a symmetric geometry with a zero rolling angle was studied, the computations were performed only for one half of the body to reduce the computation time. The computations were performed on supercomputers of the Joint Supercomputer Center (Moscow) and of the Siberian Supercomputer Center (Novosibirsk). Low-altitude computations required approximately 10^8 particles and $2 \cdot 10^7$ collision cells. The most expensive computations took about 300 processor-hours. Up to 128 processors were used. The efficiency of parallel computations was about 0.85.

Zones of extreme heating on the spacecraft surface were obtained in these computations. The bow shock wave at low altitudes (below 100 km) was found to be rather thin; when the bow shock wave is incident onto the spacecraft wing, it induces significant local heating, which can lead to disintegration of the spacecraft structure. Figure 4 shows the pressure and Mach number fields and also the surface distribution of the heat-transfer coefficient at an altitude of 95 km. Zones of extreme heating at the tip of the wing and at the area of incidence of the bow shock wave are clearly visible.

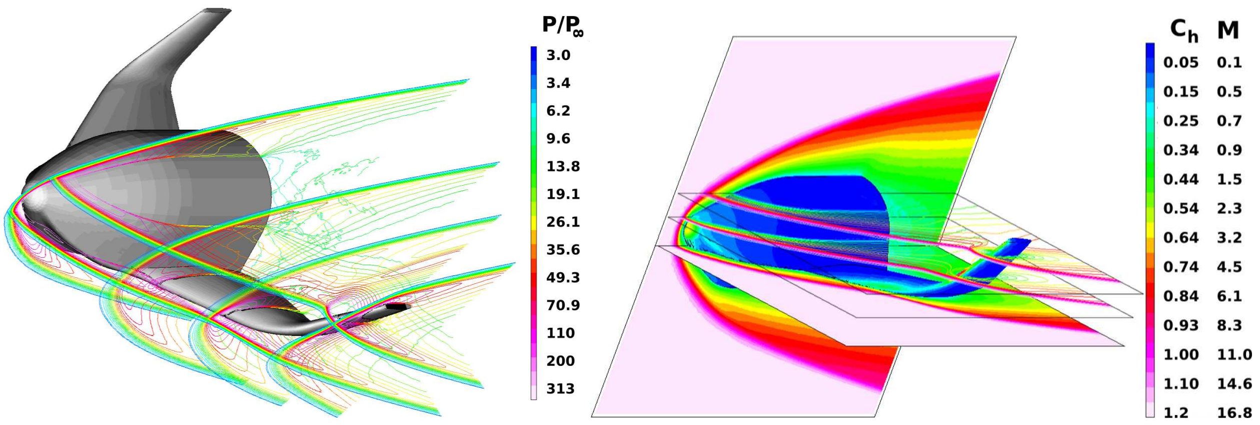

FIGURE 4. Pressure and Mach number flowfields around the Clipper spacecraft. Heat transfer coefficient distribution. Altitude 95 km.

Reentry vehicle

Another example of computations performed by the SMILE++ software system is the computation of aerothermodynamic characteristics of a promising reentry vehicle. As in the previous case, the study was aimed at obtaining aerothermodynamic parameters of the vehicle along its descent trajectory. Investigations of this problem were focused on studying the influence of chemical reactions on aerothermodynamic characteristics. The pressure flowfield around this vehicle is shown in Fig. 5. The effect of chemical reactions on distributions of thermodynamic characteristics in the region of control flaps was estimated. The drag coefficient changed only by 3%, while the heat-transfer coefficient in the case with allowance for chemical reactions decreased threefold at an altitude of 75 km (heat-transfer coefficients for different altitudes are listed in Table 2). The computations were performed on 256 processors in the Joint Supercomputer Center (Moscow, Russia). The computations took approximately 7000 processor-hours; the parallelization efficiency was about 0.77.

CONCLUSIONS

A powerful software system SMILE++ for studying rarefied gas flows by the DSMC method is presented. The system design is based on the principles of Object-Oriented Programming, which makes it readily modifiable in order to add new capabilities. The multiprocessing functionality of SMILE++ is realized through an MPI interface and is capable of running on multiprocessor SMP machines with shared memory and HPC clusters with distributed memory. The dynamic load balancing algorithms realized inside the computation code allow one to achieve high speedups and efficiency even on a large number of processors (up to 1000 and more).

The SMILE++ system provides a complete lifecycle of computations starting from a geometry model, preprocessing, going through the computation proper, and finishing with post-processing and presentation of results. All SMILE++ subsystems have a Graphic User Interface, which makes them user-friendly and easy to use. Some results of application of the system are given, which demonstrate the system capabilities for computing various problems of rarefied gas dynamics.

TABLE 2. Heat transfer coefficient C_h of the reentry vehicle.

Altitude, km	100	90	85	80	75
Nonreacting flow	0.368	0.136	0.0947	0.0622	0.043
Reacting flow	0.31	0.0618	0.0338	0.0211	0.013

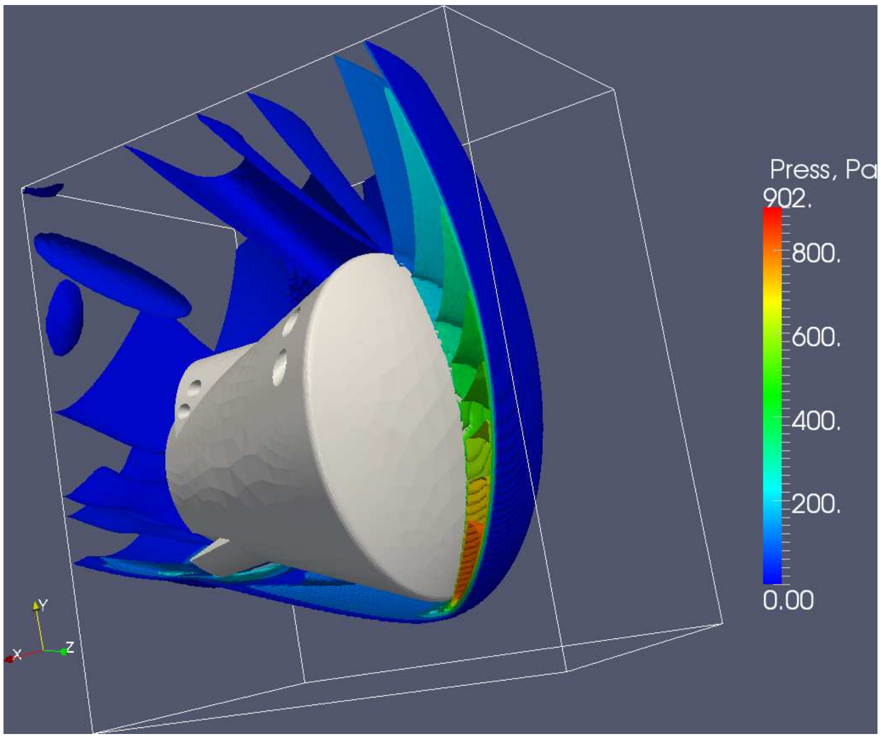

FIGURE 5. Pressure flowfield around the promising reentry vehicle. Altitude 80 km. Angle of attack 40 deg.

ACKNOWLEDGMENTS

The present study is supported by the Russian Foundation for Basic Research (RFBR Project No. 10-08-01203-a), Lavrentyev Youth Grant "High-altitude aerothermodynamics of advanced spacecraft taking into account non-equilibrium chemical reactions," and Program No. 11 of the Presidium of the Russian Academy of Sciences. This support is gratefully acknowledged.

REFERENCES

1. G.A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Clarendon Press, Oxford, 1994.
2. G.A.Bird. "The DS2V/3V Program Suite for DSMC Calculations." Rarefied Gas Dynamics: 24th Int. Symp., Porto Giardino, Italy, July 10-16, 2004, (Ed. M.Capitelli), AIP Conference proceedings, Vol. 762, 2005, pp. 541-546.
3. Dietrich, S. and Boyd, I. D., 1996. "Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method." *Journal of Computational Physics*, vol. 126, pp. 328-342.
4. G.J.LeBeau "A User Guide for the DSMC Analysis Code (DAC) Software for Simulating Rarefied Gas Dynamic Environments," Revision DAC97-G, Jan. 2002.
5. M. S. Ivanov and S. V. Rogasinsky, "Analysis of Numerical Techniques of the Direct Simulation Monte Carlo Method in the Rarefied Gas Dynamics," *Sov. J. Numer. Anal. Math. Model.* 3, 453-465 (1988).
6. M.S. Ivanov, G.N. Markelov, S.F. Gimelshein, "Statistical Simulation of Reactive Rarefied Flows: Numerical Approach and Applications." AIAA Paper 98-2669, Albuquerque, June 1998.
7. A.Kashkovsky, G.Markelov and M.Ivanov, "An Object-Oriented Software Design for the Direct Simulation Monte Carlo Method," AIAA 2001-2895.
8. 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.