computation for each volume does not depend on the computation for the rest of volumes and, therefore, this stage is performed in parallel. The minimum of all the local time steps previously obtained for each volume is computed. The $(n+1)$th state of each volume is approximated from the nth state using the data computed in the previous phases. This stage is also completed in parallel.

6 FLUX CALCULATIONS

The implementation of the finite volume method using a global memory and register file is illustrated in Fig. 4. Each time layer calculation is performed in two stages. Two kernels are used for the parallel implementation of the finite volume method on GPU, one of which calculates the flow through the faces of control volumes (stage 1, Fig. 4*a*), and the other one provides flow variables calculation on the next time layer (stage 2, Fig. 4*b*). On the first stage, flow variables in the centers of control volumes are stored in array Q in global memory. One thread is used to calculate the fluxes through the faces of control volume. Each thread uses the flow variables vector in the control volumes i and $i+1$. Fluxes through

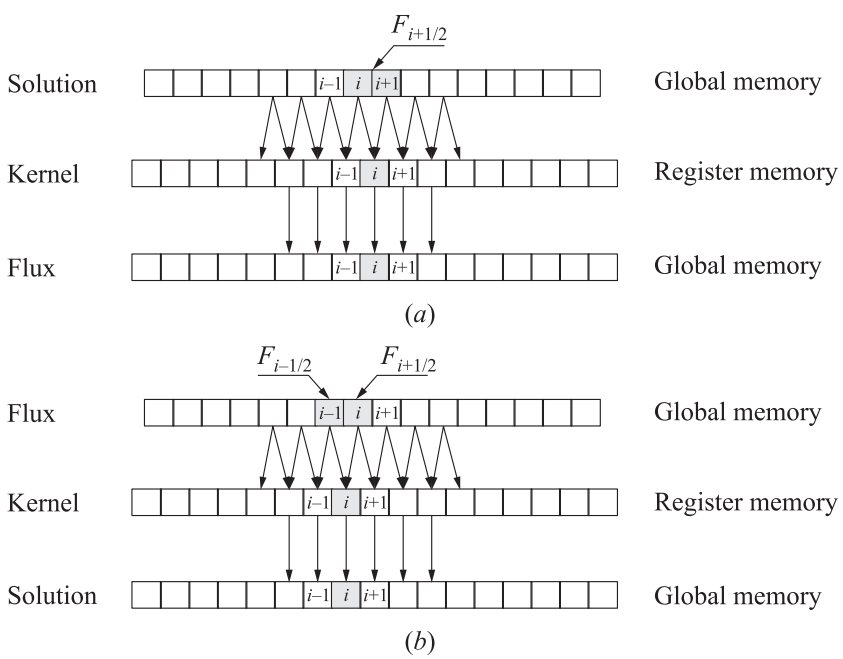

Figure 4 Flux calculation (*a*) and calculation of flow variable vector on a new time layer (*b*)

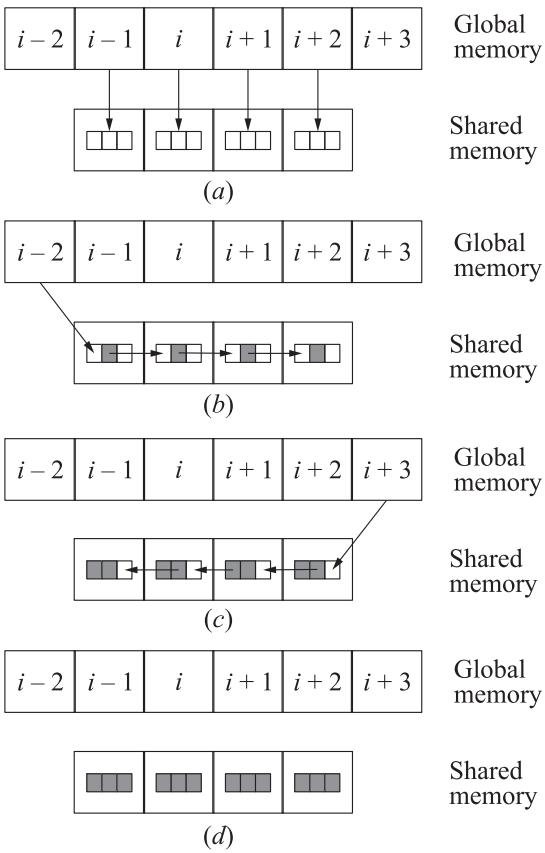

Figure 5 Use of shared memory and flux calculation

faces are stored in array F. On the second stage, a set of threads correspond to the same number of control volumes. To calculate the flow variables vector on a new time level, the fluxes through the faces $i - 1/2$ and $i + 1/2$ are used, and the solution computed in the control volume i. The solution is stored in the array Q.

The use of shared memory in the calculation of flow variable vector is presented in Fig. 5, which shows how to copy the data from global memory to shared memory. For example, the implementation of upwind-type scheme require the use of three control volumes to calculate fluxes and limiters. On step 1, flow variables vector corresponding to the centered location is copied (Fig. 5*a*), and on steps 2 and 3, flow variables vectors corresponding to the left and right locations are copied (Figs. 5*b* and 5*c*). Each thread makes treatment of the three flow variables vectors stored in the shared memory (Fig. 5*d*).

Table 1 Updating of flow variables and flux calculation

Step	Thread		
	$i-1$	i	$i+1$
Variables	u_{i-2}, u_{i-1}, u_i	u_{i-1}, u_i, u_{i+1}	u_i, u_{i+1}, u_{i+2}
Reconstruction	u^L_{i-1}, u_{i-1}, u^R_{i-1}	u^L_i, u_i, u^R_i	u^L_{i+1}, u_{i+1}, u^R_{i+1}
Restriction	u^{L*}_{i-1}, u_{i-1}, u^{R*}_{i-1}	u^{L*}_i, u_i, u^{R*}_i	u^{L*}_{i+1}, u_{i+1}, u^{R*}_{i+1}
Evolution	$\widehat{u}^L_{i-1}$, u_{i-1}, $\widehat{u}^R_{i-1}$	$\widehat{u}^L_i$, u_i, $\widehat{u}^R_i$	$\widehat{u}^L_{i+1}$, u_{i+1}, $\widehat{u}^R_{i+1}$
—		Synchronization	
Flux function	$f(\widehat{u}^R_{i-2}, \widehat{u}^L_{i-1})$	$f(\widehat{u}^R_{i-1}, \widehat{u}^L_i)$	$f(\widehat{u}^R_i, \widehat{u}^L_{i+1})$
—		Synchronization	
Flux	$f_{i-2} - f_{i-1}$	$f_{i-1} - f_i$	$f_i - f_{i+1}$

Table 1 shows the order of updating flow variables vector in the shared memory. In the simulation of two-dimensional flows (four flow variables), each thread is processing four flow vectors, and the total number of used shared memory is estimated as $4 * n_v *$ `blockDim.y` $*$ `blockDim.x`.

7 RESULTS AND DISCUSSION

The GPU version of the CFD code is tested for a variety of benchmark cases. Numerical calculations are performed with unstructured in-house finite volume CFD code. An equivalent solver is made in C++ to be run in a CPU for benchmarking purposes. All meshes are treated as unstructured meshes. This issue is related to the presentation of the meshes used in the solution of practical problems in the CFD code and not to the visual representation of the meshes.

7.1 Sod Problem

The Sod problem constitutes a particularly interesting and difficult test case, since it presents an exact solution to the full system of one-dimensional (1D) Euler equations containing simultaneously a shock wave, a contact discontinuity, and an expansion fan [14]. This problem is chosen to validate the numerical schemes and assess the temporal accuracy of the numerical solution obtained by the present method, since an analytical solution exists. The initial conditions in the present computation are the following: $\rho_L = 1.0$, $u_L = 0.75$, and $p_L = 1.0$ if $0 \leq x \leq 50$, and $\rho_R = 0.125$, $u_R = 0$, and $p_R = 1.0$ if $50 < x \leq 100$.

Table 2 Sod problem. Time and speedup

No.	Mesh 1			Mesh 2			Mesh 3			Mesh 4		
	CPU	GPU	S	CPU	GPU	S	CPU	GPU	S	CPU	GPU	S
1	1.63	0.13	12.43	47.70	0.20	245.25	460.64	0.92	502.50	4627.61	8.06	574.39
2	0.14	0.07	1.87	5.51	0.17	33.17	43.58	0.57	76.00	436.09	5.22	83.48

Calculations are performed on different meshes. A number of cells increases from 1024 cells for mesh 1 to 30,720 cells for mesh 2 and to 307,200 cells for mesh 3. The finest mesh, mesh 4, contains about three million cells. The time step is 15.2 μs, and the total calculation time is 7.63 ms. Courant number is equal to 0.85. Calculations are performed on one module of Tesla S1070 platform with 1.44 GHz (number of cores is 256) and on a single core of CPU AMD Phenom 2 with 3 GHz.

The time required for calculation of one time step, and speedup of calculations are given in Table 2 (time is given in milliseconds). Option 1 corresponds to Godunov scheme, and option 2 corresponds to Roe scheme. For both options, a good growth of speedup, S, is observed. However, Godunov method is not ideal from the parallelization point of view, since the exact solution of the Riemann problem involves a large number of data exchange, reducing the GPU performance.

Speedup and memory required are compared based on numerical solutions of different test cases. Numerical accuracy of the results obtained is similar for different meshes used in calculations. For 1D case, Godunov's scheme is less expensive from the computational point of view requiring a small number of communications between processors. In 3D calculations, numerical solution based on Godunov's method is more time consuming as compared to Roe scheme.

7.2 Shock Tube Problem

Three-dimensional model and unstructured mesh are used to solve the shock tube problem. The length of the computational domain is $L = 10$ m (Fig. 6). Calculations are based on different meshes. The coarsest mesh contains about 10^4 cells (mesh 1) and the finest mesh contains about 10^7 cells (mesh 4). The intermediate meshes contain 10^5 cells (mesh 2) and 10^6 (mesh 3) cells. The time step is 15.2 μs, and the total computational time is 7.63 ms. Courant number is equal to 0.85. The calculations are performed on one module of Tesla S1070 platform with 1.44 GHz (a number of cores is 256) and one core of CPU AMD Phenom 2 with 3 GHz.

Time of calculation of 1000 time steps and speedup of calculations are presented in Table 3 (time is given in seconds). Three indices are used to specify

computational option. The first index corresponds to the solution of Euler equations (option 1) or to the solution of Navier–Stokes equations (option 2). The second index corresponds to the time-marching scheme used in calculations based on one-step (option A) or two-step (option B) Runge–Kutta scheme. The third index corresponds to the exact (indices 1 and 3) or approximate (indices 2 and 4) Riemann solver of the first (indices 1 and 2) or second (indices 3 and 4) order. The calculations based on the finest mesh containing about 10 million of cells (mesh 4) with Godunov scheme give speedup of 42 (option 1A2). For the solution of viscous problem with the scheme of the second order, the speedup drops to 22 (option 2A2).

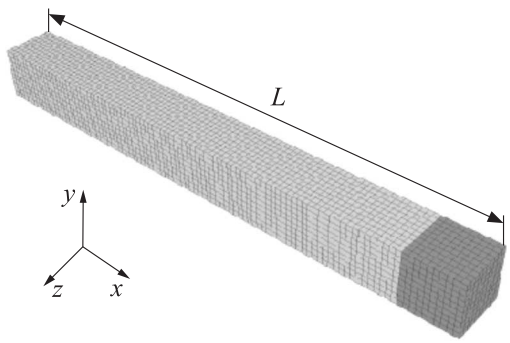

Figure 6 Shock tube problem

The time required for calculation of 1000 time steps on the mesh with 10^7 cells and memory usage are given in Table 4. Option 1 corresponds to GPU calculations based on Godunov scheme, option 2 corresponds to CPU calculations based on Godunov scheme, and option 3 corresponds to calculations based on commercial package Ansys Fluent with 8 nodes.

Table 3 Shock tube problem. Time and speedup

No.	Mesh 1			Mesh 2			Mesh 3			Mesh 4		
	CPU	GPU	S	CPU	GPU	S	CPU	GPU	S	CPU	GPU	S
1A1	6.63	0.47	13.96	69.99	2.64	26.50	671.22	22.67	29.61	6329.61	198.63	31.87
1A2	14.72	0.62	23.68	121.83	3.73	32.68	934.05	31.60	29.56	8787.27	207.82	42.28
1A3	12.03	1.29	9.30	130.33	10.41	12.52	1262.17	94.27	13.39	11018.60	826.18	13.34
1A4	20.66	1.55	13.33	193.47	12.81	15.11	1630.47	115.92	14.06	13934.40	872.42	15.97
2A1	18.79	1.36	13.83	198.96	10.98	18.12	1918.32	99.60	19.26	17485.10	874.99	19.98
2A2	27.79	1.60	17.35	261.63	13.31	19.65	2285.81	120.88	18.91	20542.40	914.83	22.46
1B1	12.63	0.91	13.95	133.88	5.07	26.39	1283.90	44.05	29.14	12163.30	382.90	31.77
1B2	29.14	1.20	24.30	242.48	7.29	33.26	1909.59	61.83	30.89	16989.20	406.65	41.78
1B3	23.47	2.50	9.37	254.60	20.39	12.49	1630.47	186.18	8.76	22484.20	1636.73	13.74
1B4	41.57	3.04	13.67	380.87	25.60	14.88	3227.67	230.86	13.98	27978.30	1734.85	16.127
2B1	36.83	2.63	13.98	388.74	21.25	18.29	3776.11	197.58	19.11	35257.70	1736.47	20.30
2B2	56.28	3.14	17.90	515.97	26.28	19.63	4672.10	240.95	19.39	40595.40	1822.97	22.27

Table 4 Shock tube problem. Time and memory

No.	Memory, MB	S/M	Time, s	S/M
1	2582.28	—	305.29	—
2	2696.72	1.04	14916.60	48.86
3	8210.16	3.18	7662.00	25.10

Graphics processing unit implementation of the CFD code is preferable against parallel implementation of similar algorithms in commercial CFD package (for the given computational resources).

7.3 Lid-Driven Cavity Problem

Large-eddy simulation of lid-driven cavity flow is considered at $\mathrm{Re} = 10^3$. The Smagorinsky model is used [15] as a subgrid scale model and a projection method is applied to discretization of the Navier–Stokes equations. The mesh contains 128^3 nodes.

Time costs are presented in Fig. 7 for various parts of the computational procedure.

Poisson equation is solved with the red/black Gauss–Seidel method (routines red_kernel and black_kernel), taking about two third of the total computation time.

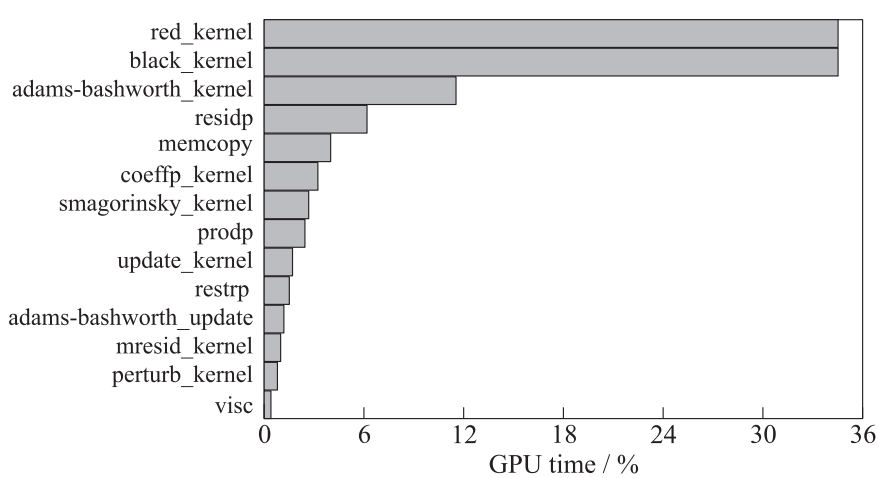

Figure 7 Time consuming for different steps

Table 5 Lid-driven cavity problem. Time and speedup

Mesh	$n_x = n_y = n_z = 4$			$n_x = 32$, $n_y = 1$, $n_z = 8$		
	CPU, s	GPU, s	S	CPU, s	GPU, s	S
16^3	0.34	0.39	0.86	0.34	0.31	1.11
32^3	3.08	1.24	2.50	3.08	0.66	4.73
64^3	31.14	6.49	4.81	31.14	2.71	11.51
128^3	291.05	50.92	5.72	291.05	18.35	15.88

The next expensive routine in terms of computational time is a routine adams_bashworth implementing time evolution.

The routine residp calculates the residuals of the Poisson equation.

Copy and memory allocation for flow variables is produced by the routine memcopy.

Routines coeffp_kernel and smagorinsky_kernel produce calculations of the coefficients related to the discrete Poisson equation and calculations based on the Smagorinsky model.

The routines update_kernel and adams_bashworth_kernel are used to go to the next time step.

Contributions of other routines such as prodp, restrp, mresid_kernel, perturb_kernel, and visc, supporting some nonimportant operations, are relatively small.

Speedup of calculations on different meshes with different partitioning techniques into blocks of size $n_x \times n_y \times n_z$ are presented in Table 5 (computational time is given in seconds for 100 time steps). On the mesh containing 128^3 nodes, speedup changes in 2.8 times for different partitioning techniques.

The most time consuming part of the computational algorithm is the solution of Poisson equation for pressure. The developed CFD solver uses a simple iterative scheme to solve Poisson equation. More advanced methods (Krylov subspace methods, multigrid methods) could be applied in the future. Preliminary results obtained show their advantages against iterative methods in terms of computational time and overall speedup (however, computational work per time step increases).

7.4 Channel Flow Problem

Large-eddy simulation of turbulent flow in a channel with square cross-sectional shape is carried out at the Reynolds number $\mathrm{Re}_\tau = 360$ on the mesh with $256 \times 64 \times 64$ nodes. The ratio of the channel length to its width is $L/H = 4$. The mesh is uniform along the x coordinate and the mesh is stretched near the walls; so, $y^+ \sim z^+ \sim 1$ on the channel walls.

Table 6 Channel flow problem. Time and speedup

Mesh	$n_x = n_y = n_z = 4$			$n_x = 32$, $n_y = 1$, $n_z = 8$		
	CPU	GPU	S	CPU	GPU	S
$256 \times 64 \times 64$	208.09	34.56	6.02	208.09	14.32	14.50
$512 \times 64 \times 64$	448.47	64.29	6.98	—	—	—

The calculations are based on the Smagorinsky subgrid scale model [15]. The nonlinear CFD solver works in an explicit time marching fashion, based on a three-step Runge–Kutta stepping procedure and piecewise parabolic method. The governing equations are solved with Chakravarthy–Osher scheme for inviscid fluxes and the central difference scheme of the 2nd order for viscous fluxes. Calculations are performed on the CPU Intel Core 2 Duo with 3 GHz and GPU card GeForce GTX 480.

Speedup of calculations on different meshes is shown in Table 6 based on different partitioning techniques. The computational time is given in seconds for the 100 time steps. On the mesh with $256 \times 64 \times 64$ nodes, changing a partitioning technique into blocks leads to speedup increasing by 2.4 times.

7.5 Flat Plate Flow

The turbulent flow over a smooth flat plate is well-known CFD benchmark solution [16], and it is used for verification and validation of other CFD codes.

The length of the computational domain is $30L$ ($10L$ before the plate and $20L$ behind the plate) and the width of the computational domain is $20L$ where L is the length of the plate ($L = 1$ m). Free stream velocity ($U = 10$ m/s), static pressure ($p = 101\,325$ Pa), and static temperature ($T = 300$ K) are fixed on the inlet boundary. No-slip and no-penetration boundary conditions are used on the plate. The plate is adiabatic. Weak boundary conditions are applied to the outlet boundary. Slip boundary conditions are used on the far-stream boundaries.

The flat plate boundary layer problem is solved on different meshes. Laminar flow calculations are performed on one core of AMD Phenom 2.3 GHz and one

Table 7 Flat plate problem. Time and speedup

Number of nodes	CPU	GPU	SSS
$1.3 \cdot 10^5$	0.140	0.003	46.67
$1.3 \cdot 10^6$	1.406	0.026	54.08
$6.6 \cdot 10^6$	7.091	0.126	56.28
$1.3 \cdot 10^7$	14.06	0.251	56.02

module of Tesla S1070 platform consisting of 240 cores with 1.44 GHz. The mesh consists of 111,670 nodes. Computational time of one iteration is 462.6 s on CPU and 11.5 s on GPU and speedup is 40.2. The turbulent flow calculations are based on CPU Xeon X5670 2.93 GHz and one module of Tesla S2050 platform. The computational time and speedup of calculations are shown in Table 7 for one iteration. Increasing a number of nodes from 10^5 to 10^7, speedup increases on 10%.

8 CONCLUDING REMARKS

Graphics processing units have evolved as a new paradigm for scientific computations. They are essentially multicore machines with a large number of computational units sharing a common memory. They are viewed as SIMD computers. Their cost/performance ratio and low power consumption make them attractive for high-resolution CFD computations. However, in order to exploit the inherent architecture of the device, the numerical algorithm as well as data structures are carefully tailored to minimize the memory access and any recursive relations in the algorithm.

Possibilities of the use of GPUs for CFD calculations are discussed. The finite volume method is applied to solve full Euler and Navier–Stokes equations on unstructured mesh. Technology CUDA is used for programming implementation of parallel computational algorithms. Solution of some CFD problems on GPUs is presented and approaches to optimization of the CFD code related to the use of different types of memory are discussed. Speedup of solution on GPUs with respect to solution on CPU is compared with the use of different meshes and different methods of distribution of input data into blocks. Speedup of CFD calculations changes from 10 to 50 depending on the problem, computational procedure, and computational resources. This makes GPUs very attractive for computing industrial fluid flows.

The computational procedure is used as a part of CFD package LOGOS developed in the Institute of Theoretical and Mathematical Physics of the Russian Federal Nuclear Center (Sarov, Russia). LOGOS package is widely used in mechanical engineering and aerospace applications.

Further work is focused on parallel implementation of implicit schemes and convergence acceleration techniques.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research (grant 13-07-12079). The authors wish to thank colleagues from the Russian Federal Nuclear Center (Sarov, Russia) for the access to the computational resources and discussion of the computational results.

REFERENCES

1. Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Krűger, A. E. Lefohn, and T. J. Purcell. 2007. A survey of general-purpose computation on graphics hardware. *Comput. Graph. Forum* 26(1):80–113.
2. Sanders, J., and E. Kandrot. 2011. *CUDA by example: An introduction to general-purpose GPU programming*. Boston: Pearson Education. 312 p.
3. Scheidegger, C. E., J. L. D. Comba, and R. D. da Cunha. 2005. Practical CFD simulations on programmable graphics hardware using SMAC. *Comput. Graph. Forum* 24(4):715–728.
4. Hagen, T. R., K.-A. Lie, and J. R. Natvig. 2006. Solving the Euler equations on graphics processing units. *Computational science*. Eds. V. N. Alexandrov, G. Dick van Albada, P. M. A. Sloot, and J. Dongarra. Lecture notes in computer science ser. Berlin–Heidelberg: Springer Verlag. 3994:220–227.
5. Brandvik, T., and G. Pullan. 2009. An accelerated 3D Navier–Stokes solver for flows in turbomachines. ASME Paper. No. GT2009-60052.
6. Thibault, J. C., and I. Senocak. 2009. CUDA implementation of a Navier–Stokes solver on multi-GPU desktop platforms for incompressible flows. AIAA Paper No. 2009-758.
7. Shinn, A. F., S. P. Vanka, and W. W. Hwu. 2010. Direct numerical simulation of turbulent flow in a square duct using a graphics processing unit (GPU). AIAA Paper No. 2010-5029.
8. Kuo, F.-A., M. R. Smith, C.-W. Hsieh, C.-Y. Chou, and J.-S. Wu. 2011. GPU acceleration for general conservation equations and its application to several engineering problems. *Comput. Fluids* 45(1):147–154.
9. Kampolis, I. C., X. S. Trompoukis, V. G. Asouti, and K. C. Giannakoglou. 2010. CFD-based analysis and two-level aerodynamic optimization on graphics processing units. *Comput. Method. Appl. Mech. Eng.* 199(9–12):712–722.
10. Corrigan, A., F. Camelli, R. Lőhner, and F. Mut. 2011. Semi-automatic porting of a large-scale Fortran CFD code to GPUs. *Int. J. Numer. Meth. Fl.* 69(2):314–331.
11. Volkov, K. N., V. N. Emelyanov, A. G. Karpenko, I. V. Kurova, A. E. Serov, and P. G. Smirnov. 2013. Numerical solution of fluid mechanics problems on general-purpose graphics processor units. *Numer. Meth. Programming* 14(1):82–90.
12. Volkov, K. N., V. N. Emelyanov, A. G. Karpenko, P. G. Smirnov, and I. V. Teterina. 2013. Implementation of a finite volume method and calculation of flows of a viscous compressible gas on graphics processor units. *Numer. Meth. Programming* 14(1):183–194.
13. Volkov, K. N. 2010. Large-eddy simulation of free shear and wall-bounded turbulent flows. In: *Atmospheric turbulence, meteorological modelling and aerodynamics*. USA: Nova Science. 505–574.
14. Sod, G. A. 1978. A survey of several finite difference methods of systems of nonlinear hyperbolic conservation laws. *J. Comput. Phys.* 27(1):1–31.
15. Smagorinsky, J. 1963. General circulation experiments with the primitive equations. *Mon. Weather Rev.* 91(3):99–164.
16. Schlichting, H., and K. Gersten. 2000. *Boundary layer theory*. Berlin: Springer Verlag. 799 p.