Development and acceleration of unstructured mesh-based CFD solver

Vladislav Emelyanov, Anton Karpenko** and Konstantin Volkov****
**Faculty of Power Engineering, Baltic State Technical University*
1, ul. 1-ya Krasnoarmeyskaya, St Petersburg, 190005, Russia
***Faculty of Mathematics and Mechanics, St Petersburg State University*
Universitetsky prospect, Old Petergof, St Petersburg, 198504, Russia
****Faculty of Science, Engineering and Computing, Kingston University*
Friars Avenue, Roehampton Vale, London, SW15 3DW, United Kingdom

Abstract

The study was undertaken as part of a larger effort to establish a common CFD code for simulation of internal and external flows, and involves some basic validation studies. The governing equations are solved with finite volume code on unstructured meshes. The computational procedure involves reconstruction of the solution in each control volume and extrapolation of the unknowns to find the flow variables on the faces of control volume, solution of Riemann problem for each face of the control volume, and evolution of the time step. The non-linear CFD solver works in an explicit time-marching fashion, based on a three-step Runge-Kutta stepping procedure. Convergence to a steady state is accelerated by the use of geometric technique, and by the application of Jacobi preconditioning for high speed flows, with a separate low Mach number preconditioning method for use with low speed flows. CFD code is implemented on GPUs. Speed-up 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. The results obtained provide promising perspective for designing a GPU-based software framework for applications in CFD.

1. Introduction

Fluid flow and heat transfer occur in nature and engineering. There are numerous naturally occurring phenomena as well as technical and technological applications in which fluid flows and heat transfer play an important role.

The methods of computational fluid dynamics (CFD) are extensively applied in design and optimization of industrial devices to get more insight into 3D unsteady flows through fluid or gas passages. Accurate prediction of compressible flows still remains a challenging task despite a lot of work in this area. The quality of CFD calculations of the flows strongly depends on the proper prediction of flow physics (shock waves, rarefaction waves, recirculation regions). Investigations of heat transfer, skin friction, secondary flows, flow separation and re-attachment effects demand reliable numerical methods, accurate programming and robust working practices.

The stagnation in the clock-speed of central processing units (CPU) has led to significant interest in parallel architectures that offer increasing computational power by using many separate processing units. Modern graphics hardware contains such an architecture in the form of the graphics processing units (GPU). GPU platforms including GPU clusters make it possible to achieve speedups of an order of magnitude over a standard CPU in many CFD applications and are growing in popularity [1].

Speed and accuracy are key factors in the evaluation of CFD solver performance. In CFD applications, the increasing demands for accuracy and simulation capabilities produce an exponential growth of the required computational resources. High performance computing (HPC) resources are widely used in engineering applications.

The use of GPUs is a cost effective way of improving the performance in CFD applications [2]. Taking advantage of any multi-core architecture requires programs to be written for parallel execution. For CFD, this has traditionally meant splitting the flow domain into several parts (domain decomposition) that are solved independently on each processor node in a cluster, with the flow properties at boundaries being communicated between the nodes after each time step (processor balancing). This is also the process adopted for GPUs, but the GPU introduces several additional constraints that make the stream programming paradigm particularly useful [3].

Although GPU has attractive characteristics for massively parallel computations, it has not been implemented in CFD for a long time due to the complex programming techniques. Developers must have special knowledge about computer graphics which is unfamiliar for general CFD researchers. Thanks to the CUDA (Compute Unified Device Architecture) library provided by NVIDIA, researchers are free from the restrictions of computer hardware knowledge and need to concentrate on CFD algorithms and CUDA programming language.

Depending on the complexity of the CFD problem to represent and solve, structured or unstructured meshes are used. Computational algorithms are more efficiently implemented on structured meshes, and data structures to handle the mesh are easy to implement [4,5]. However, structured meshes present poor accuracy if the problem to be solved has complex internal or external boundaries. On the other hand, unstructured meshes present more flexibility and higher accuracy to represent problems that have complex geometries and boundaries [6]. However, the data structures to handle it are not easy to implement, and also explicit neighboring information should be stored.

Much of the efforts in running CFD codes on GPUs has been directed toward the case of CFD solvers based on structured and block-structured meshes [3, 7–12]. These solvers are easily to implement on GPUs due to their regular memory access pattern. There are various examples of implementation of CFD solvers on structured meshes for simulation of flows of viscous incompressible fluid [13–15].

Unstructured mesh based analysis methods on HPC systems with shared memory and distributed memory have been largely studied. However, shared and distributed memory systems are fundamentally different from GPUs. A GPU is a SIMT (Single Instruction Multiple Thread) engine, whereas shared and distributed memory systems are MPMD (Multiple Program Multiple Data) engines. The common aspect of these parallel engines is that in both of them the mesh application is limited by memory latency. Achieving good performance for unstructured mesh based CFD solvers on GPUs is more difficult due to their data dependent and irregular memory access patterns [16–18].

The most of the work done so far has either been for relatively small codes written from scratch or for a small portion of a large existing code. However, GPU support is available in mathematical packages (MATLAB) and commercial CFD solvers (ANSYS CFX, ANSYS Fluent).

Multigrid and preconditioning techniques are widely used to increase performance of the CFD code. A key issue is effective implementation of these techniques on unstructured meshes.

The present work is undertaken as a part of a larger effort to establish a common CFD code for simulation of flows in aerospace and mechanical applications, and involves some basic validation studies. Up to now, a few researches on fully 3D compressible Navier–Stokes GPU solver for engineering applications have been reported. The motivation of this paper is to assess the in-house compressible CFD code, and to demonstrate successful design of a highly parallel computation system based on GPUs and validate the speedup factor compared with CPU. The code is programmed following the standard of CUDA C language. Single precision arithmetic is kept through the entire residual computations with the help of latest GPU hardware and careful design of CFD code. The results obtained are generally in reasonable agreement with the available experimental and computational data reported in literature. The parallelization methods are studied and speedup factor by GPU cards is measured.

2. Governing equations

In Cartesian coordinates (x, y, z), an unsteady 3D flow is described by the following equation written in conservative form

$$\frac{\partial Q}{\partial t} + \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} = 0. \tag{1}$$

The pressure is calculated as

$$p = (\gamma - 1)\rho \left[e - \frac{1}{2}\left(v_x^2 + v_y^2 + v_z^2\right) \right].$$

The vector of conservative variables, Q, and the flux vectors, F_x, F_y and F_z, have the following form

$$Q = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ \rho e \end{pmatrix},$$

$$F_x = \begin{pmatrix} \rho v_x \\ \rho v_x v_x + p - \tau_{xx} \\ \rho v_x v_y - \tau_{xy} \\ \rho v_x v_z - \tau_{xz} \\ (\rho e + p) v_x - v_x \tau_{xx} - v_y \tau_{xy} - v_z \tau_{xz} + q_x \end{pmatrix},$$

$$F_y = \begin{pmatrix} \rho v_y \\ \rho v_y v_x - \tau_{yx} \\ \rho v_y v_y + p - \tau_{yy} \\ \rho v_y v_z - \tau_{yz} \\ (\rho e + p) v_y - v_x \tau_{yx} - v_y \tau_{yy} - v_z \tau_{yz} + q_y \end{pmatrix},$$

$$F_z = \begin{pmatrix} \rho v_z \\ \rho v_z v_x - \tau_{zx} \\ \rho v_z v_y - \tau_{zy} \\ \rho v_z v_z + p - \tau_{zz} \\ (\rho e + p) v_z - v_x \tau_{zx} - v_y \tau_{zy} - v_z \tau_{zz} + q_z \end{pmatrix}.$$

The components of viscous stress tensor and components of heat flux vector are found from the relations

$$\tau_{ij} = \mu_e \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \frac{2}{3} \frac{\partial v_k}{\partial x_k} \delta_{ij} \right), \quad q_i = -\lambda_e \frac{\partial T}{\partial x_i}.$$

Here, t is the time, ρ is the density, v_x, v_y, and v_z are the velocity components in the coordinate directions x, y, and z, p is the pressure, e is the total energy per unit mass, T is the temperature, and γ is is the ratio of specific heat capacities.

The governing equations written in the form (1) are suitable for both laminar and turbulent flows. In simulation of turbulent flows, the effective viscosity, μ_e, is calculated as a sum of molecular viscosity, μ, and eddy viscosity, μ_t, and the effective thermal conductivity, λ_e, is expressed in terms of viscosity and Prandtl number. They are expressed as

$$\mu_e = \mu + \mu_t, \quad \lambda_e = c_p \left(\frac{\mu}{\mathrm{Pr}} + \frac{\mu_t}{\mathrm{Pr}_t} \right),$$

where c_p is the specific heat capacity at constant pressure. Molecular and turbulent Prandtl numbers are $\mathrm{Pr} = 0.72$ and $\mathrm{Pr}_t = 0.9$ for air. The Sutherland's law is used to obtain molecular viscosity as a function of temperature

$$\frac{\mu}{\mu_*} = \left(\frac{T}{T_*} \right)^{3/2} \frac{T_* + S_0}{T + S_0},$$

where $\mu_* = 1.68 \times 10^{-5}$ kg/(m s), $T_* = 273$ K and $S_0 = 110.5$ K for air.

3. Numerical method

The non-linear CFD solver works in an explicit time-marching fashion, based on a Runge–Kutta stepping procedure. The flux vector is split into the inviscid and viscous components. The governing equations are solved with upwind finite difference scheme for inviscid fluxes, and central difference scheme of the second order for viscous fluxes. The unstructured CFD code developed uses an edge-based data structure to give the flexibility to run on meshes composed of a variety of cell types [19]. The fluxes through the surface of a cell are calculated on the basis of flow variables at nodes at either end of an edge, and an area associated with that edge (edge weight). The edge weights are pre-computed and take into account geometry of the cell.

In conservative variables, the equation describing an unsteady viscous compressible gas flow is written as

$$\frac{\partial Q}{\partial t} + \nabla \cdot \Big[F(Q) + G(Q, \nabla Q)\Big] = 0, \tag{2}$$

where Q is the vector of conservative variables, $F(Q)$ is the vector of inviscid fluxes, and $G(QU, \nabla Q)$ is the vector of viscous fluxes.

Integrating equation (2) over the control volume V with boundary ∂V, whose orientation is specified by the outer unit normal $\boldsymbol{n}$, and applying the Gauss–Ostrogradsky theorem, gives

$$\frac{\partial}{\partial t}\int\limits_V Q\, d\Omega + \int\limits_{\partial V} F(Q)\cdot \boldsymbol{n}\, dS + \int\limits_{\partial V} G(Q, \nabla Q)\cdot \boldsymbol{n}\, dS = 0. \tag{3}$$

The governing equation (3) solved by the CFD code is of the form

$$\frac{dQ}{dt} = R(Q), \tag{4}$$

where Q is the flow variables vector averaged over the control volume. The flow residual is

$$R(Q) = S(Q) - L(Q),$$

where $L(Q)$ denotes all the spatial differencing terms, and $S(Q)$ denotes terms from boundary conditions and possible source terms.

Equation (4) is written in the form

$$\frac{dQ_i^n}{dt} + L(Q_i^n) = 0, \tag{5}$$

where $L(Q_i^n)$ is the differential operator. The subscript i refers to the control volume, and the superscript n refers to the time layer.

The three-step Runge–Kutta method is used for discretization of the equation (5) in time

$$Q_i^{(1)} = Q_i^{(0)} + \Delta t L(Q_i^{(0)});$$

$$Q_i^{(2)} = \frac{3}{4}Q_i^{(0)} + \frac{1}{4}\left[Q_i^{(1)} + \Delta t L(Q_i^{(1)})\right];$$

$$Q_i^{(3)} = \frac{1}{3}Q_i^{(0)} + \frac{2}{3}\left[Q_i^{(2)} + \Delta t L(Q_i^{(2)})\right].$$

Here, $Q_i^0 = Q_i^{(n)}$ and $Q_i^{(3)} = Q_i^{(n+1)}$.

The inviscid flux is found from the relation

$$F(Q_L, Q_R) = \frac{1}{2}\Big[F(Q_L) + F(Q_R) - |A|\,(Q_R - Q_L)\Big].$$

where the subscripts L and R refer to cells on the left and on the right edges of the control volume. The matrix A is presented in the form $A = R\Lambda L$, where Λ is the diagonal matrix composed from the Jacobian eigenvalues, and R and L are the matrices composed from its right and left eigenvectors, respectively.

The time step is found from the estimation of the inviscid and viscous fluxes

$$\frac{1}{\Delta t_i} = \frac{1}{\mathrm{CFL}} \max\left\{\frac{1}{\Delta t_i^1}, \frac{\beta}{\Delta t_i^2}\right\},$$

where CFL is the Courant–Friedrichs–Lewy number, and $\beta \sim 0.5$. The time step Δt_i^1 is calculated on the basis of the spectral radius of the Jacobian of the discrete inviscid operator, and the time step Δt_i^2 is found on the basis of the quasi-linear form of viscous fluxes written in primitive variables and the spectral radius of the Jacobian of the discrete viscous operator.

Convergence to a steady state is accelerated by the use of multigrid techniques [20], and by the application of block-Jacobi preconditioning for high-speed flows, with a separate low-Mach number preconditioning method for use with low-speed flows [21, 22]. The sequence of meshes is created using an edge-collapsing algorithm.

The computational procedure is implemented as a computer code in C/C++ programming language. Parallelization of the computational procedure is performed by a message passing interface (MPI). CUDA technology is used to implement GPU version of the code.

4. Preconditioning

Numerical methods for compressible gas equations that perform well at moderately subsonic and supersonic flow velocities become low effective or unsuitable as applied to flows at low Mach numbers ($\mathrm{M} < 0.2$), which is manifested by slower convergence of time marching to a steady state and by the loss of accuracy of the resulting steady state solutions. The slower convergence of time marching is explained by the fact that the stiffness of the compressible Euler and Navier–Stokes equations increases as $\mathrm{M} \to 0$ (this feature is exhibited at the differential level). The stiffness is characterized by the ratio of the maximum to minimum eigenvalues of the Jacobian (the ratio of the maximum to minimum propagation velocities of perturbations). The integration time step is determined by the velocity of the fastest wave (acoustic waves, $\lambda = |u+c|$), while the time required for reaching a steady state depends on the velocity of the slowest wave (convective waves, $\lambda = |u|$). In viscous problems and turbulent flow computations on stretched meshes in boundary layers, the time step is restricted by the acoustic solution modes and by the mesh size in the normal direction to the wall.

Preconditioning makes it possible to eliminate the stiffness of the original system and to accelerate the convergence of time marching to a steady state. Additionally, subsonic flows can be computed more accurately by applying a modified discretization of convective fluxes in the preconditioned equations. In the general case, preconditioning changes the form of the underlying equations and the properties of difference schemes because it introduces artificial viscosity.

A preconditioning method is developed. It makes it possible to construct a universal numerical procedure for computing inviscid and viscous compressible flows in a wide range of Mach numbers (from essentially subsonic to transonic and supersonic flow velocities). The preconditioning matrix is constructed by applying the approach proposed in [23]. This approach relies on physical variables (one of which is temperature). Its features include a specific form of writing fluxes, the computation of a dissipative term in the course of finding the fluxes through control volume faces, and a specific representation of matrices in the diagonalization of the inviscid flux Jacobian of the preconditioned system.

5. Multigrid method

The multigrid method is not a fixed technique but rather a stencil and a build-up construction whose implementation efficiency depends on the adaptation of its components to the problem in question.

The multigrid cycle consists of the following steps: pre-smoothing, residual calculation at the current mesh level, restriction and correction of the residual on the coarse mesh, prolongation and interpolation of the error to a fine mesh, correction of the fine mesh solution using the correction interpolated from the coarse mesh (coarse mesh correction), and post-smoothing for the suppression of the high-frequency error components appearing after the interpolation to the fine mesh. The computations terminate when a prescribed accuracy is achieved.

The smoothing method (smoother) damps the high-frequency error modes and appears the element of the multigrid method that is most dependent on the type of the problem. The role of the smoother is not so much to reduce the total error but rather to smooth it (suppress the high frequencies) so that the error admits a good approximation on the coarse mesh. Standard smoothers are linear iterative methods (Jacobi, Gauss–Seidel, and incomplete factorization methods).

The quality of the multigrid method is determined by the chosen sequence of meshes and interpolation operator. Quality criteria include the convergence factor, which shows how fast the method converges (how many iteration steps are required to achieve the prescribed level of the residual) and the complexity of the restriction and prolongation operators, which determines the number of operations and the amount of storage required for each iteration step.

The implementation of the multigrid approach is illustrated in the Figure 1 (V-cycle). The iterations begin with the mesh of the highest resolution (level 1). The number of mesh levels is nlevel. The arrows indicate the transmission of data from one mesh level to another. The horizontal arrows show that the computations (smoothing) occurs at a single mesh level. The CFL number is estimated by executing nstart iteration steps (one iteration step is used by default). The parameter npre specifies the number of iterations used for pre-smoothing at each mesh level (in the transition from a fine to a coarse mesh). The parameter npost defines the number of iterations used for postsmoothing (in the transition from a coarse to a fine mesh). The parameter ncrs specifies the number of iterations executed on the coarsest mesh.

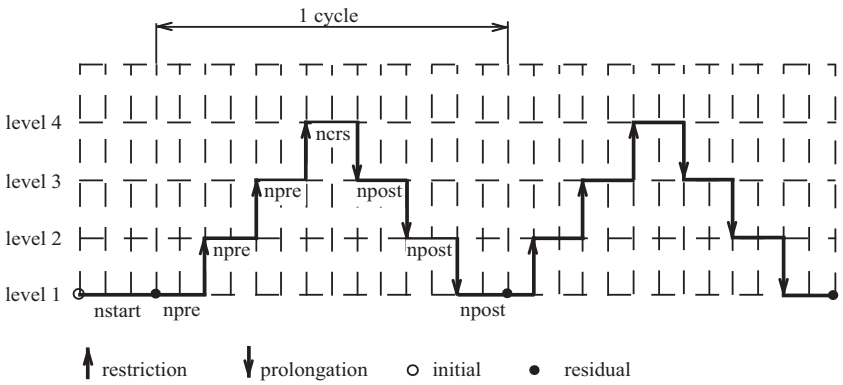

Figure 1: Implementation of the multigrid method (V-cycle)

A multigrid method is proposed for solving the Euler and Navier–Stokes equations on an unstructured mesh. The multigrid technique producing a sequence of meshes via collapsing faces is implemented using a structure associated with mesh faces (edge weights). The discretization of the equations on a sequence of nested meshes is easy to implement, since the finite volume method developed has no constrains on the number of cells, their faces, or and the cell shape (for hybrid meshes, the cell shape in the original mesh changes in the transition to the next mesh level).

6. Parallel implementation

The performance of critical portion of the CFD solver consists of a loop which repeatedly computes the time derivatives of the conserved variables. The conserved variables are then updated using an explicit Runge–Kutta time-stepping procedure. The most expensive computation consists of accumulating flux contributions across each face when computing the time derivatives. Therefore, the performance of the CUDA kernel which implements this computation is crucial in determining whether or not high performance is achieved.

The finite volume mesh is generated from input data with the appropriate setting of initial and boundary conditions. The computation steps required by the problem considered are classified into two groups, computations associated to faces and edges, and computations associated to volumes. The numerical scheme exhibits a high degree of data parallelism because the computation at each edge/volume is independent with respect to the computation performed at the rest of edges/volumes. Moreover, the explicit scheme presents a high arithmetic intensity and the computation exhibits a high degree of locality.

The implementation is split between CPU and GPU. Pre- and post-processing steps are done on the CPU, leaving only the computation itself to be performed on the GPU. For example, the CPU constructs the mesh and evaluates the face areas, face normals and cell volumes. The initialization of the flowfield is also done on the CPU. Each time step of the computation then involves a series of kernels on the GPU which evaluate the cell face fluxes, sum the fluxes into the cell, calculate the change in properties at each node, smooth the variables and apply boundary conditions. Each kernel operates on all the nodes (no distinction is made between boundary nodes and interior nodes). This causes difficulties if an efficient code is to be obtained. For example, the change in a flow property at a node is formed by averaging the flux sums of the adjacent cells (for mesh with quadrangle cells, four cells surround an interior node, but only two at a boundary node). This problem is overcome using dependent texturing. The indices of the cells required to update a node are pre-computed on the CPU and loaded into GPU texture memory. For a given node, the kernel obtains the indices required and then looks up the relevant flux sums which are stored in a separate GPU texture. This avoids branching within the kernel.

The implementation of the finite volume method using a global memory and register file is illustrated in the Figure 2. Each time layer calculations are 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), and the other one provides flow variable calculations on the next time layer (stage 2). On the first stage, flow variables in the centers of control volumes are stored in global memory (array Q). One thread is used to calculate the fluxes through the faces of control volume. Each thread uses the flow variables vector in adjacent control volumes, i and $i+1$. Fluxes through cell faces are stored in array F. On the second stage, a set of threads corresponding to the same number of control volumes is launched 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 is computed in the control volume i. The solution is then stored in the array Q.

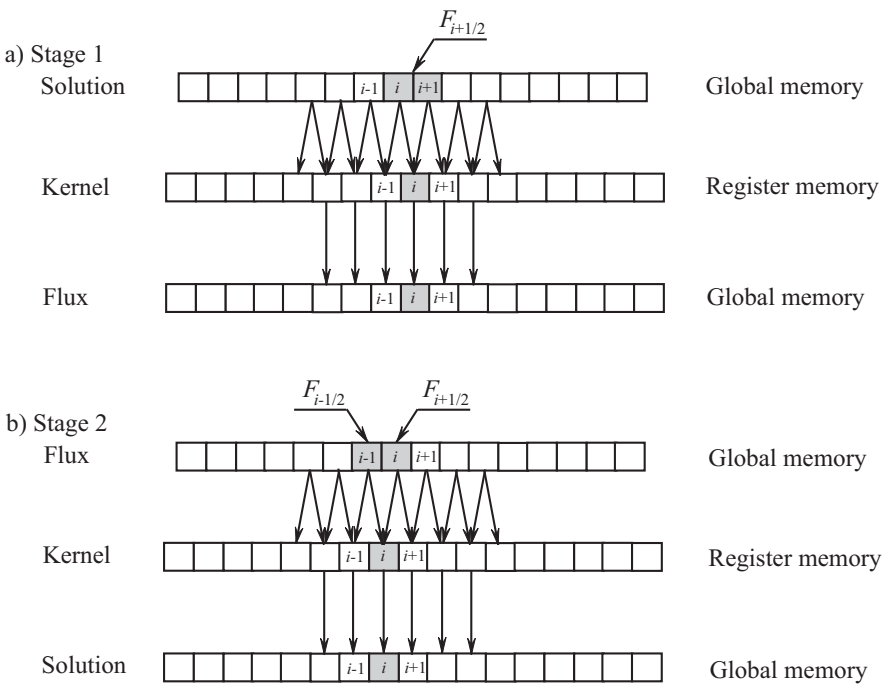

Figure 2: Flux calculation (a) and calculation of flow variables vector on a new time layer (b)

The most costly stage in the algorithm is edge-based calculations involving two calculations for each face communicating two cells. This contribution is computed independently for each face and is added to the partial sums associated to each cell. For each control volume, the local time step is computed. The 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 control volume is approximated from the n-th state using the data

computed in the previous phases. This stage is also completed in parallel.

7. Test cases

A series of model simulations in a wide range of Mach numbers are used to analyze the convergence rate and accuracy of steady state solutions of the original and preconditioned gas dynamics equations. These equations are integrated until a steady state solution is reached.

7.1 Flow through a channel with a bump

The flow through a plane channel with a bump is considered. The length to height ratio in the channel is $L/H = 4$, and the maximum height of the bump (which is a circular arc) is $0.1H$ (the maximum bump is 10% of the channel width). The computations are performed on a mesh of 120×20 cells (Figure 3) with 60 nodes placed on the bump surface.

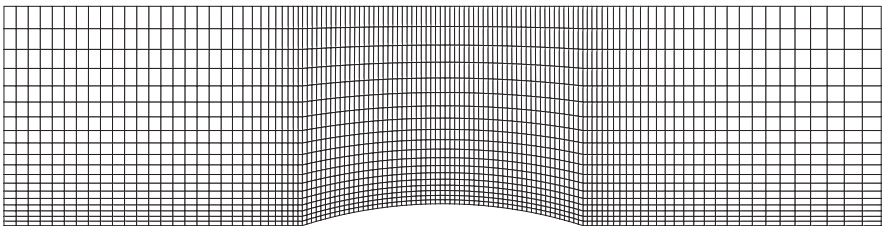

Figure 3: Computational mesh

Flows through a channel with a bump are computed, for example, in [24, 25]. Specifically, the implicit Euler time differencing and the Beam–Warming scheme for discretizing inviscid fluxes are used in [24]. The computations in [25] are based on Godunov's method and are performed in a wide range of Mach numbers.

The velocity ($U = 3.47$ m/s), pressure ($p = 10^5$ Pa), and temperature ($T = 300$ K) are applied to the inlet cross section of the channel, while mild boundary conditions (free outflow) are specified in the outlet cross section. The inlet cross section conditions corresponded to M = 0.01. A steady state solution of the problem is obtained by taking 5000 time steps of the time marching procedure.

Figure 4 displays level lines of the velocity magnitude. In contrast to the solution of the original equations, a velocity distribution symmetric about the vertical axis is obtained in the case of preconditioning.

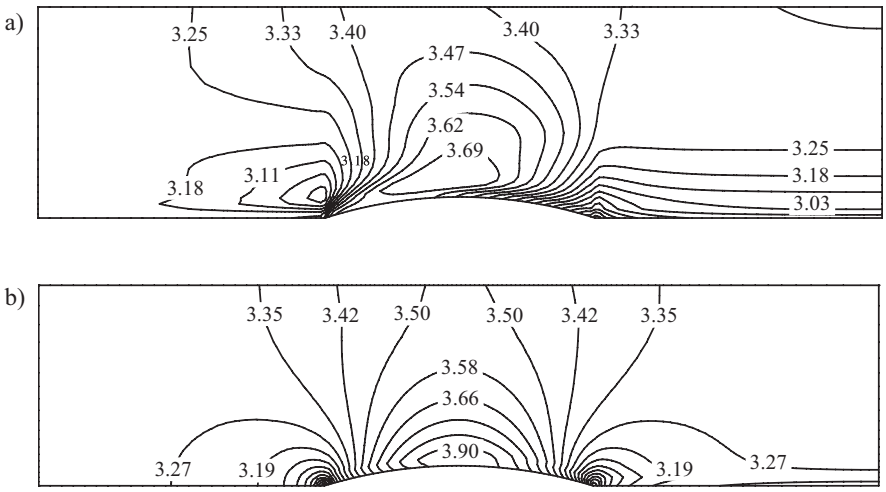

Figure 4: Contours of the velocity magnitude in the case of the original (a) and preconditioned (b) equations

The convergence rate of the time marching procedure is shown in the Figure 5. The original equations are solved in conservative variables, while the preconditioned equations are computed in physical variables.

Lines 1 and 2 depict the residuals (in physical variables) caused by discretizing the momentum equation, while line 3 shows the residual caused by discretizing the pressure equation. In the case of preconditioning, the prescribed residual is obtained after about 3500 iteration steps. For the original equations, the residuals with respect to velocity and pressure are two orders of magnitude higher and the convergence rate nearly ceases to vary after 4000 time steps.

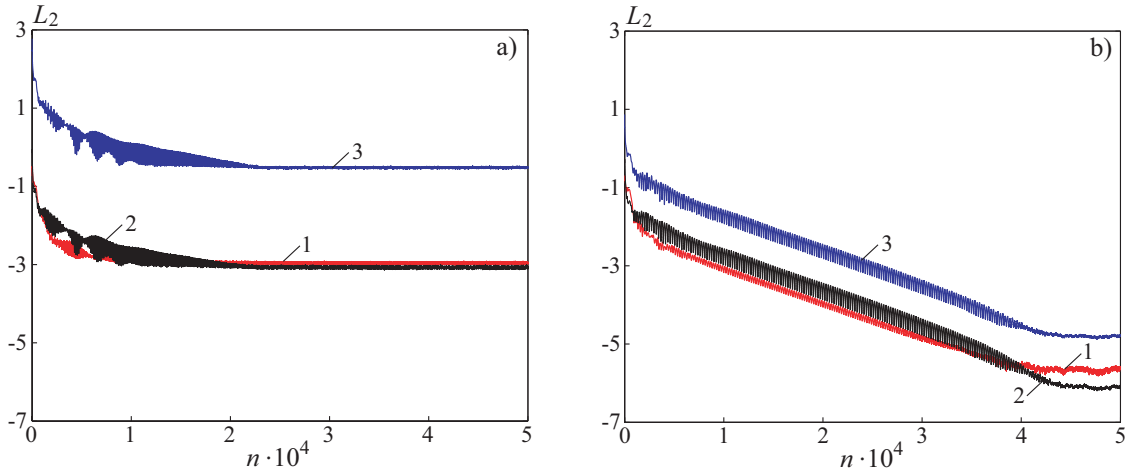

Figure 5: Convergence rate of the solutions of the original (a) and preconditioned (b) equations

To test the performance and accuracy of the numerical method in a wide range of Mach numbers, the computations are performed in subsonic, transonic, and supersonic regimes. More specifically, the flow is computed in a channel with a 10% bump (as in the underlying version) on a mesh of 144×32 cells at $\mathrm{M} = 0.5$ (subsonic) and $\mathrm{M} = 0.675$ (transonic) and in a channel with a 4% bump on a mesh of 220×60 cells at $\mathrm{M} = 1.65$ (supersonic).

For subsonic and supersonic regimes, Figure 6 shows level lines of the velocity magnitude at various inlet Mach numbers. For relatively low inlet Mach numbers, the flow is nearly symmetric about the vertical axis (Figure 6a). The weak asymmetry of the flow is associated with the leading and trailing edges of the bump (a horseshoe vortex of weak intensity develops behind the bump). To eliminate these shortcomings, the flow characteristics near the corner points are computed by interpolating the flow parameters from interior nodes of the computational domain [25]. At high inlet Mach numbers, shock waves develop and interact in the flow (Figure 6b). The inclination angles of the shocks and the level lines agree well with the numerical data presented in [25].

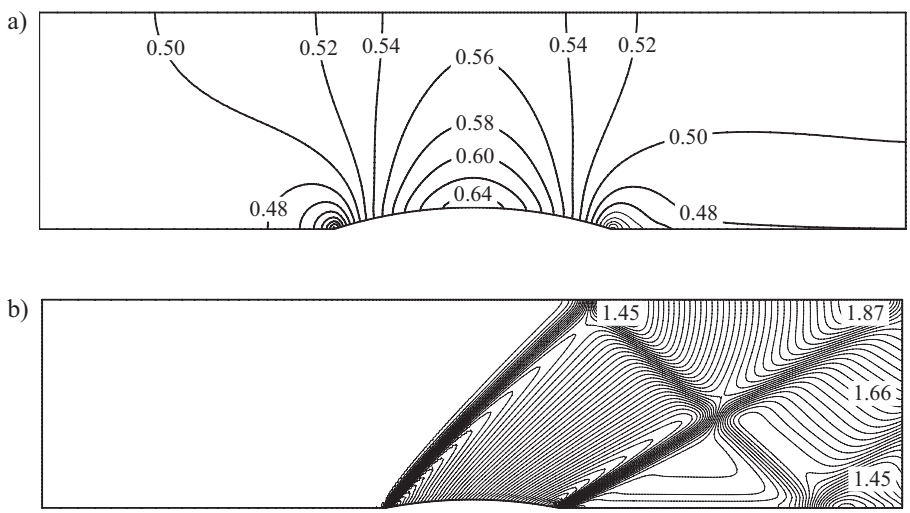

Figure 6: Contours of the velocity magnitude at $\mathrm{M} = 0.5$ (a) and $\mathrm{M} = 1.65$ (b)

Figure 7 presents the Mach number distributions on the upper (line 1) and lower (line 2) walls of the channel in various flow regimes. These distributions agree well with numerical data from [34] (as in the case of velocity magnitude level lines, weak differences are observed on the lower wall of the channel near the corner points).

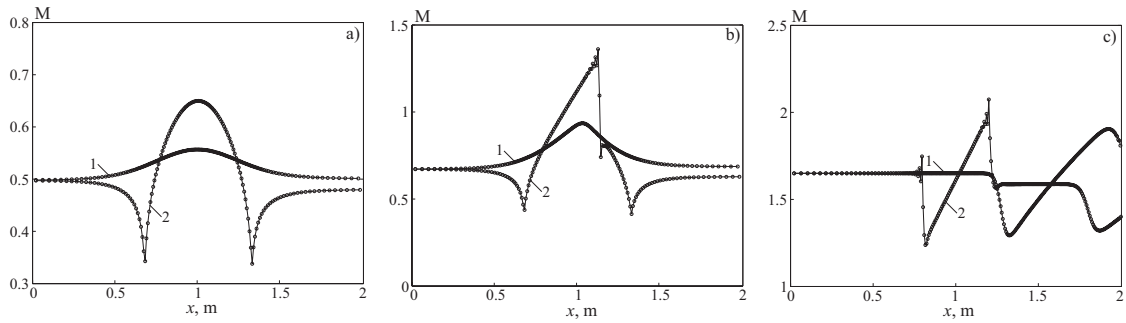

Figure 7: Mach number distribution along the upper (line 1) and lower (line 2) channel walls at M = 0.5 (a), M = 0.675 (b), and M = 1.65 (c)

The numerical results obtained in the test problem suggest that the numerical method developed has a sufficient accuracy for resolving characteristic features of incompressible and compressed flows. Due to the preconditioning procedure, the convergence rate of time marching is made independent of the Mach number. At low Mach numbers, the CPU time required for solving the preconditioned equations is more by about 15% (due to an increase in the number of arithmetic operations) than in the case of the original equations.

7.2 Flow over airfoil

Transonic flow of viscous compressible gas over RAE2822 airfoil is considered at the angle of attack of 2.4°. The length of the computational domain is $50L$ ($15L$ before the airfoil and $25L$ behind the airfoil), and the width of the computational domain is $20L$, where L is the cord length ($L = 1$ m). Unstructured triangle mesh contains 38265 cells (Figure 8). A number of time steps is 10^4.

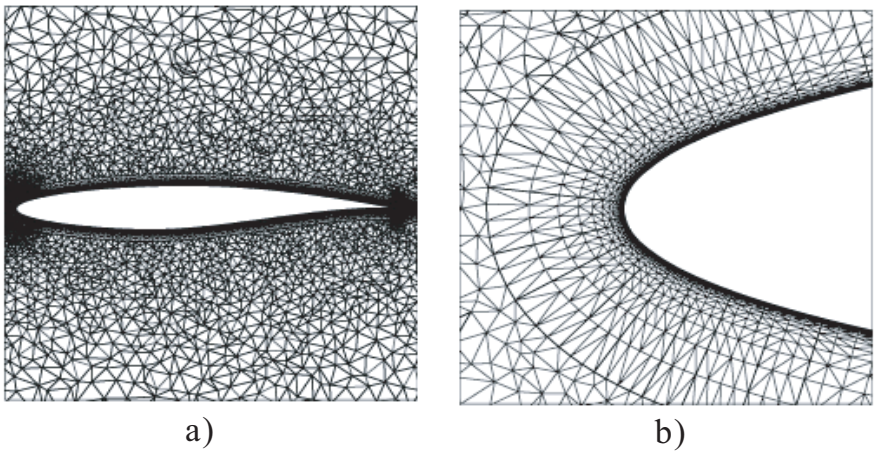

Figure 8: Mesh over airfoil (a) and mesh near airfoil (b)

Free stream boundary conditions (Mach number is fixed at $M_\infty = 0.725$) are used on the inlet boundary. No-slip and no-penetration boundary conditions are applied to airfoil. Airfoil is treated as adiabatic boundary. Zero-gradient boundary conditions are applied to the outlet boundary. Slip boundary conditions are specified on the top and bottom boundaries. Periodic boundary conditions are used in spanwise direction.

In the computations, we used four mesh levels and the mesh generation technique based on cells collapsing in the direction of the shortest face (semi-coarsening method), which makes it possible to capture the boundary layer on the airfoil.

The computations are performed on an unstructured triangular mesh consisting of 11 298 nodes (version 1a) and on a hybrid mesh consisting of 19 126 nodes (version 1b). A structured mesh is used near the airfoil. In the case of a hybrid mesh, the number of cells is 24339 in the mesh of level 1 (10 692 triangles and 13647 quadrilaterals), 11484 in the mesh of level 2 (5527 triangles and 5957 quadrilaterals), 5528 in the mesh of level 3 (2795 triangles and 2733 quadrilaterals), and 2894 in the mesh of level 4 (1599 triangles and 1295 quadrilaterals). For the meshes of levels 1.4, the domain $\Delta y/L < 5 \times 10^{-6}$ contained 50, 25, 13, and 6 cells, respectively.

The aspect ratio for mesh levels 2–4 is 1:2 away from the airfoil, while near the airfoil, it is somewhat smaller than 1:2, since some of the quadrilaterals turned into triangles when a sequence of nested meshes is generated. The maximum aspect ratio is 1 for the mesh of level 1 (1 for both triangles and quadrilaterals), 0.47 for the mesh of level 2 (0.54 for triangles and 0.44 for quadrilaterals), 0.47 for the mesh of level 3 (0.51 for triangles and 0.45 for quadrilaterals), and 0.52 for the mesh of level 4 (0.57 for triangles and 0.48 for quadrilaterals).

Contours of the velocity magnitude are presented in the Figure 9. Pressure coefficient distributions over airfoil are shown in the Figure 10. The results computed are compared with those presented in [26]. Integration pressure and wall shear stresses over airfoil surface gives drag and lift coefficient ($C_x = 0.8388$ and $C_y = 0.0197$), which are in a good agreement with the experimental data.

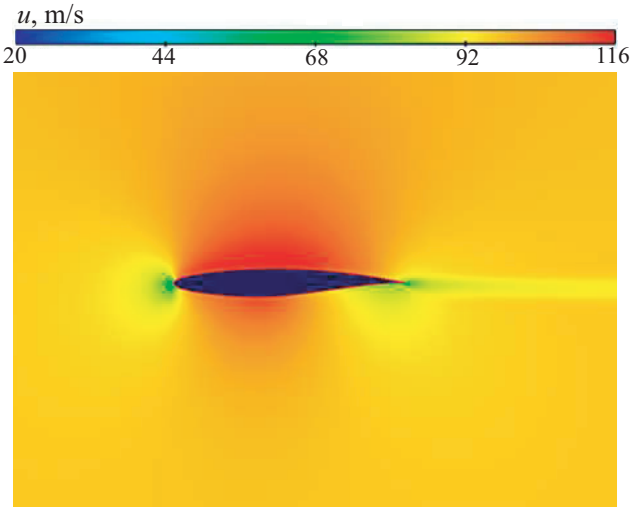

Figure 9: Contours of velocity magnitude

Distribution of the friction coefficient over airfoil, shown in the Figure 11, is in a good agreement with the data presented in [26].

Table 1 demonstrates the convergence of the iterative process on the initial segment of the residual (from 10^0 to 10^{-4}) and in its entire range (from 10^0 to 10^{-8}) (here, the residual norm is normalized by its initial value). The numerator and the denominator correspond to scalar preconditioning and Jacobi block preconditioning, respectively. The number of iterations for pre- and post-smoothing is set equal to 1. Five smoothing iterations are executed on the coarsest mesh.

Table 1: Convergence of iteration process

Option	Convergence $10^0 \to 10^{-4}$		Convergence $10^0 \to 10^{-8}$	
	Number of cycles	Time, s	Number of cycles	Time, s
1a	311/122	839/341	947/240	2522/649
1b	234/78	1563/534	1788/918	11956/6152

Figure 12 shows the convergence factor of the multigrid method as a function of the number of mesh levels for version 1a. As the number of mesh levels rises from 1 to 4, the numbers of multigrid cycles

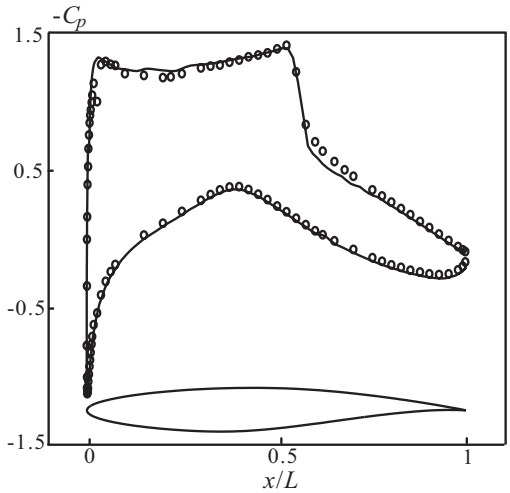

Figure 10: Comparison of the computed pressure coefficient distribution (solid line) over airfoil with the data [26] (symbols)

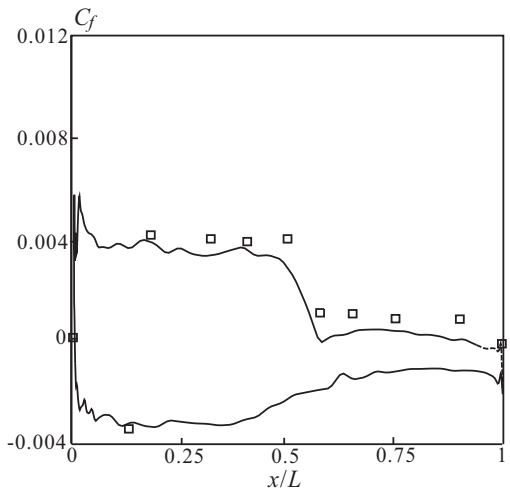

Figure 11: Comparison of friction coefficient over airfoil with data [26] (symbols)

decreases from 800 to 220. For $n = 1$, the residual reaches an approximately constant level $R \sim 10^{-5}$ after 1000 multigrid cycles (a further increase in the number of iterations does not reduce the residual), while, for $n = 4$, the residual monotonically decreases to $R \sim 10^{-8}$ beyond a small initial segment.

7.3 Speedup factor

The GPU version of the CFD code is used and validated for a variety of benchmark test 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.

The benchmark problems include flat plate boundary layer (problem A1), flow over wedge at Mach number of 2 (problem A2), flow around NACA0012 airfoil (problem A3), flow around RAE2822 airfoil (problem A4), flow around conus at Mach number of 1.6 (problem A5), flow and heat transfer behind back-forward step (problem T1), lib-driven cavity flow and heat transfer (problem T2). Speedup factors of the problems A2, A5, T1 and T2 are presented in the paper for reference only.

Laminar flow calculations of flat plate boundary layer are performed on one core of AMD Phenom 2.3

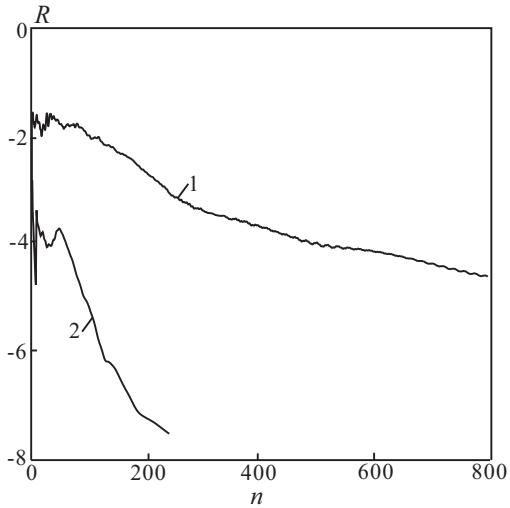

Figure 12: Residual as a function of the number of multigrid cycles in the case of (line 1) one mesh and (line 2) four nested meshes

GHz and one module of Tesla S1070 platform consisting of 240 cores with 1.44 GHz. The mesh consists of 111670 nodes. Computational time of one iteration is 462.6 s on CPU and 11.5 s on GPU, and speedup is 40.2.

Computational time of solution of one iteration on CPU and GPU and speedup are shown in the Table 2. Calculations are performed on one core of AMD Phenom 2.3 GHz and one module of Tesla S1070, consisting of 240 cores with 1.44 GHz. Speedup factor changes from 22 (problems A4 and A5) to 60 (problem T1).

Table 2: Computational time and speedup

Problem	Number of nodes	Time on CPU, s	Time on GPU, s	Speedup
A1	111670	462.6	11.5	40.2
A2	95000	288.8	12.3	23.5
A3	30836	68.1	1.6	42.6
A4	38265	141.9	6.3	22.5
A5	36333	130.8	5.8	22.4
T1	138003	295.3	4.9	59.7
T2	9801	39.9	1.8	22.6

The flat plate turbulent boundary layer problem is solved on various meshes. The turbulent flow calculations are based on CPU Xeon X5670 2.93 GHz and one module of Tesla S2050 platform. The computational time in seconds and speedup of calculations are shown in the Table 3 for one iteration. Increasing a number of nodes from 10^5 to 10^7, speedup increases on 10%.

Table 3: Time and speedup for flat plate problem

Number of nodes	CPU	GPU	S
$1.3 \cdot 10^5$	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

Speedup of solution on GPUs with respect to the solution on central processor unit (CPU) is compared. Performance measurements show that numerical schemes developed achieve 20 to 50 speedup on GPU hardware compared to CPU reference implementation. The results obtained provide promising perspective for designing a GPU-based software framework for applications in CFD.

8. Conclusion

A numerical method was developed for computing steady inviscid and viscous compressible gas in a wide range of Mach numbers. The accuracy and convergence rate of the method are independent of the Mach number. The original and preconditioned equations are discretized by applying the finite volume method on an unstructured mesh. An explicit scheme is used for time differencing, while the inviscid and viscous fluxes are discretized with the help of second order accurate schemes. Preconditioning is switched on depending on the local Mach number or the local pressure field (specifically, the preconditioned equations are always solved for the incompressible fluid model).

A multigrid method was developed for solving the Euler and Navier–Stokes equations on unstructured and hybrid meshes. The method relies on a modified form of the restriction operator and involves the generation of a sequence of nested meshes via collapsing faces (semi-coarsening method). The method for generating meshes of different levels accurately takes into account the features of the problem (the boundary layer on the wall), preserves the topology of the original mesh, and produces high quality meshes in the near wall domain (reasonable stretching and obliqueness of the cells). The capabilities of the approach are demonstrated by computing inviscid and viscous compressible flows around an airfoil on meshes of various types and dimensions. The multigrid method, in conjunction with Jacobi block preconditioning, produce a prescribed level of the residual after considerably fewer iteration steps than in the case of scalar preconditioning.

Modern graphics processing units (GPU) provide architectures and new programming models that enable to harness their large processing power and to design computational fluid dynamics (CFD) simulations at both high performance and low cost. Possibilities of the use of GPUs for the simulation of external and internal flows on unstructured meshes are discussed. The finite volume method is applied to solve three-dimensional unsteady compressible Euler and Navier–Stokes equations on unstructured meshes with high resolution numerical schemes. CUDA technology is used for programming implementation of parallel computational algorithms. Solutions of some benchmark test cases on GPUs are reported, and the results computed are compared with experimental and computational data. Approaches to optimization of the CFD code related to the use of different types of memory are considered.

Acknowledgements

This work was partially supported by the Russian Foundation for Basic Research (project 13-07-12079). The author wishes to thank colleagues from the Russian Federal Nuclear Center (Sarov, Russia) for access to high performance computing 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. *Computer Graphics Forum*. 26(1):80–113.

[2] Jacobsen, D.A., and I. Senocak. 2013. Multi-level parallelism for incompressible flow computations on GPU clusters. *Parallel Computing*. 39(1):1–20.

[3] Thibault, J.C., and I. Senocak. 2009. CUDA implementation of a Navier–Stokes solver on multi-GPU desktop platforms for incompressible flows. *AIAA Paper*. 2009-758.

[4] Fu, L., Z. Gao, K. Xu, and F. Xu. 2014. A multi-block viscous flow solver based on GPU parallel methodology. *Computers and Fluids*. 95:19–39.

[5] Tuttafesta, M., G. Colonna, and G. Pascazio. 2013. Computing unsteady compressible flows using Roe's flux-difference splitting scheme on GPUs. *Computer Physics Communications*. 184(6):1497–1510.

[6] Mavriplis, D.J. 2007. Unstructured mesh discretizations and solvers for computational aerodynamics. *AIAA Paper*. 2007-3955.

[7] Scheidegger, C.E., J.L.D. Comba, and R.D. da Cunha. 2005. Practical CFD simulations on programmable graphics hardware using SMAC. *Computer Graphics Forum*. 24(4):715–728.

[8] Hagen, T.R., K.-A. Lie, and J.R. Natvig. 2006. Solving the Euler equations on graphics processing units. *Lecture Notes in Computer Science*. 3994:220–227.

[9] Brandvik, T., and G. Pullan. 2009. An accelerated 3D Navier–Stokes solver for flows in turbomachines. *ASME Paper*. GT2009-60052.

[10] 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*. 2010-5029.

[11] 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. *Computers and Fluids*. 45(1):147–154.

[12] Fu, L., Z. Gao, K. Xu, and F. Xu. 2014. A multi-block viscous flow solver based on GPU parallel methodology. *Computers and Fluids*. 95:19–39.

[13] Appleyard, J., and D. Drikakis. 2011. Higher-order CFD and interface tracking methods on highly-parallel MPI and GPU systems. *Computers and Fluids*. 46(1):101–105.

[14] Meng, J., and K. Skadron. 2011. A performance study for iterative stencil loops on GPUs with ghost zone optimizations. *International Journal of Parallel Program*. 39(1):115–142.

[15] Krotkiewski, M., and M. Dabrowski. 2013. Efficient 3D stencil computations using CUDA. *Parallel Computing*. 39(10):533–548.

[16] 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. *Computer Methods in Applied Mechanics and Engineering*. 199(9–12):712–722.

[17] Corrigan, A., F. Camelli, R. Lőhner, and F. Mut. 2011. Semi-automatic porting of a large-scale Fortran CFD code to GPUs. *International Journal for Numerical Methods in Fluids*. 69(2):314–331.

[18] Emelyanov, V.N., A.G. Karpenko, and K.N. Volkov. 2013. Development of advanced CFD tools and their application to simulation of internal turbulent flows. *5th European Conference for Aeronautics and Space Sciences*.

[19] Volkov, K.N. 2008. Unstructured-grid finite-volume discretization of the Navier–Stokes equations based on high-resolution difference schemes. *Computational Mathematics and Mathematical Physics*. 48(7):1181–1202.

[20] Volkov, K.N. 2010. Multigrid techniques as applied to gas dynamic simulation on unstructured meshes. *Computational Mathematics and Mathematical Physics*. 50(11):1837–1850.

[21] Volkov, K.N. 2009. Preconditioning of the Euler and Navier–Stokes equations in low-velocity flow simulation on unstructured grids. *Computational Mathematics and Mathematical Physics*. 49(10):1789–1803.

[22] Volkov, K.N., and A.G. Karpenko. 2015. Preconditioning of gas dynamics equations in compressible gas flow computations at low Mach numbers. *Computational Mathematics and Mathematical Physics*. 55(6):1051–1067.

[23] Weiss, J.M., and W.A. Smith. 1995. Preconditioning applied to variable and constant density flows. *AIAA Journal*. 33(11):2050–2057.

[24] Choi, D., and C.L. Merkle. 1985. Application of time iterative schemes to incompressible flow. *AIAA Journal*. 23(10):1518–1524.

[25] Eidelman, S., P. Colella, and R.P. Shreeve. 1984. Application of the Godunov method and its second order extension to cascade flow modeling. *AIAA Journal*. 22(11):1609–1615.

[26] Cook, P.H., M.A. McDonald, and G.N. Firmin. 1979. Aerofoil RAE2822 — pressure distribution and boundary layer and wake measurements. AGARD Advisory Report AR-138.