ISSN 0965-5425, Computational Mathematics and Mathematical Physics, 2013, Vol. 53, No. 8, pp. 1183–1194. © Pleiades Publishing, Ltd., 2013.
Original Russian Text © P.B. Bogdanov, A.V. Gorobets, S.A. Sukov, 2013, published in Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 2013, Vol. 53, No. 8, pp. 1360–1373.

Adaptation and Optimization of Basic Operations for an Unstructured Mesh CFD Algorithm for Computation on Massively Parallel Accelerators

P. B. Bogdanov[a], A. V. Gorobets[b], and S. A. Sukov[b]

[a] *Scientific Research Institute for System Studies, Russian Academy of Sciences, Nakhimovskii pr. 36-1, Moscow, 117218 Russia*

[b] *Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Miusskaya pl. 4, Moscow, 125047 Russia*

e-mail: bogdanov@niisi.msk.ru, andrey.gorobets@gmail.com, office@keldysh.ru

Received February 11, 2013; in final form, March 13, 2013

Abstract—The design of efficient algorithms for large-scale gas dynamics computations with hybrid (heterogeneous) computing systems whose high performance relies on massively parallel accelerators is addressed. A high-order accurate finite volume algorithm with polynomial reconstruction on unstructured hybrid meshes is used to compute compressible gas flows in domains of complex geometry. The basic operations of the algorithm are implemented in detail for massively parallel accelerators, including AMD and NVIDIA graphics processing units (GPUs). Major optimization approaches and a computation transfer technique are covered. The underlying programming tool is the Open Computing Language (OpenCL) standard, which performs on accelerators of various architectures, both existing and emerging.

DOI: 10.1134/S0965542513080046

Keywords: gas dynamics, parallel computations, unstructured meshes, GPU, OpenCL.

1. INTRODUCTION

At present, the growth of the performance of supercomputers is achieved by increasing the number of computational modules and by applying massively parallel accelerators. In the former case, algorithms of progressively higher degrees of parallelism and scalability are required, while in the latter case, algorithms have to be adapted to a different architecture and an even more complicated parallel model is required that combines various types of parallelism, namely, MIMD (Multiple Instruction Multiple Data) and SIMD (Single Instruction Multiple Data). Associated with significant modifications and sophistication of computing system architectures, this tendency necessitates considerable rearrangements in numerical algorithms and their implementations.

Among the world's most powerful supercomputers, we can see several systems of hybrid architecture, such as the Cray Titan (GPU NVIDIA Kepler); Tianhe-1A, Nebulae, TSUBAME 2.0 (GPU NVIDIA Fermi); and the Intel Stampede with Xeon Phi coprocessors of MIC (Many Integrated Core) architecture. Hybrid computations with massively parallel accelerators used as mathematical coprocessors comprise a relatively new area that emerged several years ago. Relevant programming tools and computational technologies are under active development, and the functional capabilities of the CUDA and OpenCL programming interfaces are being constantly supplemented.

The transfer of algorithms to graphics processing units (GPUs), which differ considerably from central CPUs, is a rather complicated problem. For example, matrix-vector multiplication for sparse matrices of arbitrary structure obtained in discretization on unstructured meshes is considered in [1]. This operation is a major one in iterative methods intended for systems of linear algebraic equations (SLAEs) arising in problems with unstructured discretization of the computational domain. Comprehensive optimization approaches applied to this, in fact, simple operation give an idea of the complexity of working with GPUs. The results presented in [2] for various sparse matrices suggest that even highly optimized implementations can achieve only a few percent of the peak performance of GPU devices for such operations with low specific computational costs per data item and with irregular memory access.

Nevertheless, GPUs are becoming increasingly more popular in computational fluid dynamics (CFD) and computational mathematics overall. For example, a review of various numerical GPU applications based on the CUDA programming model can be found in [3]. In gas dynamics simulation, GPUs are most

widely used in computations based on lattice Boltzmann methods (LBM). Examples of LBM algorithms can be found, for example, in [4–6]. The explicit nature of algorithms and simplicity of the data structure inherent in these methods is consistent with the GPU architecture.

An example of a finite-difference CFD algorithm designed for computations on GPU clusters with the use of MPI and CUDA was presented in [7], where rather high performance and parallel efficiency was achieved on a relatively large number of GPUs. In [8] another multi-GPU algorithm for structured grids was described, which also demonstrated considerable benefits of using accelerators. However, in all the cited works, computations were implemented on CUDA for NVIDIA GPUs, which does not allow using other accelerator architectures, such as AMD GPUs or Intel Xeon Phi GPUs of MIC architecture.

In this paper, we consider a high-order accurate finite volume algorithm on unstructured hybrid grids, which is much more complicated for GPU implementation. Spatial discretization is based on a polynomial reconstruction scheme with variables specified at the centers of grid elements. A similar approach to spatial discretization is described, for example, in [9]. The underlying programming tool is the OpenCL standard, which performs on accelerators of various architectures, both existing and emerging.

The original parallel algorithm for CPU has two-level parallelization adapted to supercomputer architecture with multicore processors with tens of thousands of cores. At the first level, MPI is used to join the nodes in the framework of a distributed memory model. At the second level, OpenMP (Open Multiprocessing) is used in the framework of a shared memory model for parallelization over the processor cores.

This paper deals with the transfer and adaptation of the basic operations in the above algorithm to the architecture of massively parallel accelerators, which are used at the third parallelization via OpenCL. The optimization of the algorithmic operations in the AMD and NVIDIA architectures is examined in detail. Two operations are discussed that are most computationally expensive and complicated from the point of view of GPU implementation, namely, the determination of polynomial reconstruction coefficients and the computation of fluxes through control volume faces. The features of the software implementation and performance-enhancing approaches are described in detail. This paper focuses only on computational optimization and does not address the design of communication schemes for MPI and CPU-GPU exchanges with overlapping computations and data transfer. This is a separate rather complicated problem to be addressed elsewhere.

This paper is organized as follows. The underlying mathematical model, spatial discretization, and the numerical method used are described in Section 2. The implementation of the algorithmic operations for the GPU architecture is covered in Section 3. The performance characteristics obtained for NVIDIA and AMD GPUs are discussed in Section 4. Finally, the main results of this work are summarized in Section 5.

2. MATHEMATICAL MODEL AND NUMERICAL IMPLEMENTATION

Below, we describe a numerical algorithm for gasdynamic simulation on unstructured meshes. This algorithm is used to address performance-enhancing strategies for GPU computations.

2.1. Mathematical Model

We consider the numerical simulation of inviscid compressible gas flows described by the Euler equations

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_1(\mathbf{Q})}{\partial x} + \frac{\partial \mathbf{F}_2(\mathbf{Q})}{\partial y} + \frac{\partial \mathbf{F}_3(\mathbf{Q})}{\partial z} = 0, \tag{1}$$

where $\mathbf{Q} = (\rho, \rho u, \rho v, \rho w, E)^{\mathrm{T}}$ is the vector of conservative gasdynamic variables. The convective fluxes $\mathbf{F}_1(\mathbf{Q})$, $\mathbf{F}_2(\mathbf{Q})$, and $\mathbf{F}_3(\mathbf{Q})$ are defined as

$$\mathbf{F}_1(\mathbf{Q}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u w \\ u(E+p) \end{pmatrix}, \quad \mathbf{F}_2(\mathbf{Q}) = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v w \\ v(E+p) \end{pmatrix}, \quad \mathbf{F}_3(\mathbf{Q}) = \begin{pmatrix} \rho w \\ \rho u w \\ \rho v w \\ \rho w^2 + p \\ w(E+p) \end{pmatrix}. \tag{2}$$

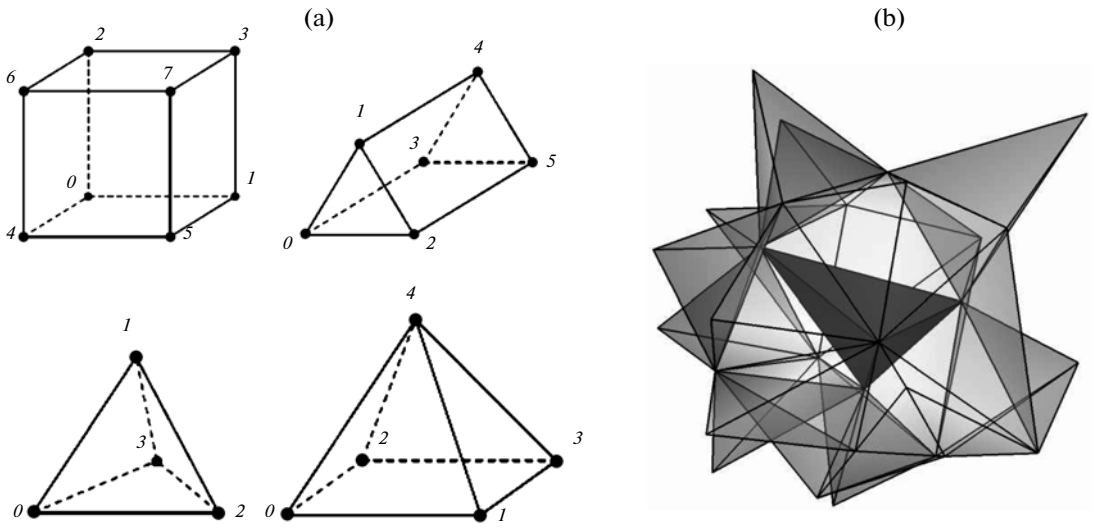

Fig. 1. (a) Types of cells and (b) example of a stencil for the polynomial reconstruction scheme.

Initially, the conservative gasdynamic variables in all grid cells are set equal to identical values corresponding to the unperturbed flow parameters $\mathbf{Q}_\infty$.

In formula (2), $\mathbf{u} = (u, v, w)$ are the Cartesian velocity components, ρ is the density, p is the pressure, $E = \rho(u^2 + v^2 + w^2)/2 + \rho\varepsilon$ is the total energy, and ε is internal energy. System (1), (2) is made closed by adding the perfect gas law $p = \rho\varepsilon(\gamma - 1)$, where γ is the ratio of specific heats.

2.2. Spatial Discretization and Numerical Method

In the framework of the finite volume method, the computational domain of the problem is divided into cells, which are equivalent to one of four types of polyhedra: a tetrahedron, hexahedron, quadrilateral pyramid, or triangular prism (Fig. 1a). Sequential numbering of elements is introduced inside the resulting unstructured hybrid mesh. Each of N_E grid elements is uniquely determined by its unique index ID, which belongs to the numerical interval $[0; N_E - 1]$. Below, we introduce notation for the geometric parameters of cells and their faces necessary in the computation:

V_{ID} is the volume of a cell;

$(xc_{ID}, yc_{ID}, zc_{ID})$ are the coordinates of the barycenter of a cell;

NS_{ID} is the number of cell faces;

I_k^{ID}, $k \in [0, NS_{ID} - 1]$, are the indices of the grid elements sharing a face with the element ID;

$\frac{ID1}{ID2}$ the face shared by the cells with indices $ID1$ and $ID2$;

$\overrightarrow{n_{\frac{ID1}{ID2}}} = \left(nx_{\frac{ID1}{ID2}}, ny_{\frac{ID1}{ID2}}, nz_{\frac{ID1}{ID2}}\right)$ is the area vector of the face $\frac{ID1}{ID2}$ codirectional with the outward normal to the cell $ID1$;

$\left(mx_{\frac{ID1}{ID2}}, my_{\frac{ID1}{ID2}}, mz_{\frac{ID1}{ID2}}\right)$ are the coordinates of the barycenter of the face $\frac{ID1}{ID2}$.

The integral-averaged values of conservative gasdynamic variables are assumed to be defined in grid elements by the formula

$$f_{ID} = \frac{\int_{V_{ID}} f(x, y, z)dV}{V_{ID}}, \tag{3}$$

where f_{ID} is the value of one of the following five variables: ρ_{ID}, ρu_{ID}, ρv_{ID}, ρw_{ID}, or E_{ID}. In the case of first-order accurate explicit time-stepping, the gasdynamic variables $\mathbf{Q}_{ID}^{n+1}$ at the time level $n-1$ are calculated using the formula

$$\mathbf{Q}_{ID}^{n+1} = \mathbf{Q}_{ID}^{n} + \frac{\Phi_{ID}\tau}{V_{ID}},$$

where τ is the current time integration step. The convective flux Φ_{ID} in a cell is the sum of the convective fluxes through its faces:

$$\Phi_{ID} = \sum_{k=0}^{NS_{ID}-1} \Phi S_{ID/I_k^{ID}}.$$

In turn, the convective flux $\Phi S_{ID/I_k^{ID}}$ through the face ID/I_k^{ID} is computed using the Roe scheme (see [10]) (F_{ROE}) and depends on the geometric parameters of the face and the values of the conservative gasdynamic variables to the left ($\mathbf{QL}_{ID/I_k^{ID}}$) and right ($\mathbf{QR}_{ID/I_k^{ID}}$) of its barycenter:

$$\Phi S_{ID/I_k^{ID}} = F_{\mathrm{ROE}}\left(QL_{ID/I_k^{ID}}, QR_{ID/I_k^{ID}}, \overrightarrow{n_{ID/I_k^{ID}}}\right). \tag{4}$$

Assuming that the values of the variables inside a cell do not vary, we have the equalities $\mathbf{QL}_{ID/I_k^{ID}} = \mathbf{Q}_{ID}$ and $\mathbf{QR}_{ID/I_k^{ID}} = \mathbf{Q}_{I_k^{ID}}$ and formula (4) becomes

$$\Phi S_{ID/I_k^{ID}} = F_{\mathrm{ROE}}\left(\mathbf{Q}_{ID}, \mathbf{Q}_{I_k^{ID}}, \overrightarrow{n_{ID/I_k^{ID}}}\right),$$

which corresponds to first-order accuracy in space.

The order of accuracy of the spatial approximation can be increased by using polynomial reconstruction of gasdynamic variables inside grid cells. The behavior of the function f_{ID} (3) inside a cell is described by a quadratic polynomial of the form

$$\begin{aligned} P_{f_{ID}}(x,y,z) &= C_{f_{ID}}^0((x-xc_{ID})^2 - g_{ID}^0) + C_{f_{ID}}^1((y-yc_{ID})^2 - g_{ID}^1) \\ &+ C_{f_{ID}}^2((z-zc_{ID})^2 - g_{ID}^2) + C_{f_{ID}}^3((x-xc_{ID})(y-yc_{ID}) - g_{ID}^3) \\ &+ C_{f_{ID}}^4((x-xc_{ID})(z-zc_{ID}) - g_{ID}^4) + C_{f_{ID}}^5((y-yc_{ID})(z-zc_{ID}) - g_{ID}^5) \\ &+ C_{f_{ID}}^6(x-xc_{ID}) + C_{f_{ID}}^7(y-yc_{ID}) + C_{f_{ID}}^8(z-zc_{ID}) + f_{ID}. \end{aligned}$$

Here, g_{ID}^h, $h \in [0, 5]$, are six constant coefficients denoting the cell mean of the elementary polynomials $(x-xc_{ID})^{\alpha_1}(y-yc_{ID})^{\alpha_2}(z-zc_{ID})^{\alpha_3}$, $\alpha_1+\alpha_2+\alpha_3 \le 3$, whose values depend on the shape of the cell ID, while $C_{f_{ID}}^l$ ($l = 1, \ldots, 8$) are the polynomial coefficients to be determined.

In the case of polynomial extension, the values of the variables at the face center with coordinates $(mx_{ID/I_k^{ID}}, my_{ID/I_k^{ID}}, mz_{ID/I_k^{ID}})$ can be computed using the polynomial reconstruction of the variables in adjacent cells:

$$\Phi S_{ID/I_k^{ID}} = F_{\mathrm{ROE}}\left(P_{\mathbf{Q}_{ID}}(mx_{ID/I_k^{ID}}, my_{ID/I_k^{ID}}, mz_{ID/I_k^{ID}}), P_{\mathbf{Q}_{I_k^{ID}}}(mx_{ID/I_k^{ID}}, my_{ID/I_k^{ID}}, mz_{ID/I_k^{ID}}), \overrightarrow{n_{ID/I_k^{ID}}}\right).$$

In the case of a quadratic polynomial, this increases the order of accuracy of the convective flux in space.

An algorithm for determining the polynomial coefficients $C^l_{f_{ID}}$ is as follows. In the initialization of the geometric parameters of the problem, a spatial stencil consisting of NG_{ID} elements with indices G^{ID}_m, $m \in [0, NG_{ID} - 1]$ is constructed around each grid element (Fig. 1b). The stencil construction method is not of key importance. For example, it can be constructed by combining elements from several neighboring levels within a dual graph, or elements lying within a sphere of given radius centered at the barycenter of the given element, etc. However, an important point is that the number of stencil elements and their relative topological positions are not defined in the general case and can differ for each element.

After a number of preliminary transformations associated with setting up an overdetermined SLAE and its solution by the least squares method, we obtain a matrix $\mathbf{A}_{ID}$ and the determination of the polynomial coefficients $C^l_{f_{ID}}$ for each gasdynamic variable is reduced to the multiplication of a matrix by a vector:

$$\begin{pmatrix} A_{ID_{0,0}} & A_{ID_{0,1}} & \dots & A_{ID_{0,NG_{ID}-1}} \\ A_{ID_{1,0}} & A_{ID_{1,1}} & & A_{ID_{1,NG_{ID}-1}} \\ A_{ID_{2,0}} & A_{ID_{2,1}} & & A_{ID_{2,NG_{ID}-1}} \\ & & & \\ A_{ID_{8,0}} & A_{ID_{8,1}} & \dots & A_{ID_{8,NG_{ID}-1}} \end{pmatrix} \begin{pmatrix} f_{G^{ID}_0} - f_{ID} \\ f_{G^{ID}_1} - f_{ID} \\ f_{G^{ID}_2} - f_{ID} \\ \dots \\ f_{G^{ID}_{NG_{ID}-1}} - f_{ID} \end{pmatrix} = \begin{pmatrix} C^0_{f_{ID}} \\ C^1_{f_{ID}} \\ C^2_{f_{ID}} \\ C^3_{f_{ID}} \\ C^4_{f_{ID}} \\ C^5_{f_{ID}} \\ C^6_{f_{ID}} \\ C^7_{f_{ID}} \\ C^8_{f_{ID}} \end{pmatrix}. \tag{5}$$

The elements of A_{ID} remain unchanged in the course of the computation. The fluxes through cell faces belonging to the boundary of the computational domain are computed according to one of the following two variants of boundary conditions: (1) free boundary (inflow or outflow boundaries) and (2) rigid wall. In the first case, the cell containing the face is assumed to be far away from the body surface; therefore, the behavior of the flow in this area can be assumed to roughly correspond to the unperturbed flow $\mathbf{Q}_\infty = (\rho_\infty, \rho u_\infty, \rho v_\infty, \rho w_\infty, E_\infty)^{\mathrm{T}}$. Accordingly, the flux through the face ID/∞ can be calculated using the formula

$$\Phi S_{ID/\infty} = F_{\mathrm{ROE}}(P_{\mathbf{Q}_{ID}}(mx_{ID/\infty}, my_{ID/\infty}, mz_{ID/\infty}), \mathbf{Q}_\infty, \overrightarrow{n_{ID/\infty}}).$$

The flux through the cell face ID/W lying on the body surface is determined by the condition that the normal velocity to the boundary surface at the barycenter of the face is set equal to zero:

$$\Phi S_{ID/W} = \begin{pmatrix} 0 \\ p_{ID}(mx_{ID/W}, my_{ID/W}, mz_{ID/W}) \cdot nx_{ID/W} \\ p_{ID}(mx_{ID/W}, my_{ID/W}, mz_{ID/W}) \cdot ny_{ID/W} \\ p_{ID}(mx_{ID/W}, my_{ID/W}, mz_{ID/W}) \cdot nz_{ID/W} \\ 0 \end{pmatrix}.$$

As initial conditions in the problem, identical values of the conservative gasdynamic variables corresponding to the unperturbed flow $\mathbf{Q}_\infty$ are set in all the grid cells.

3. IMPLEMENTATION OF THE ALGORITHMIC OPERATIONS

The above numerical method can be formalized in the form of a block diagram as shown in Fig. 2. From the point of view of software implementation, the most complicated stages (namely, the computa-

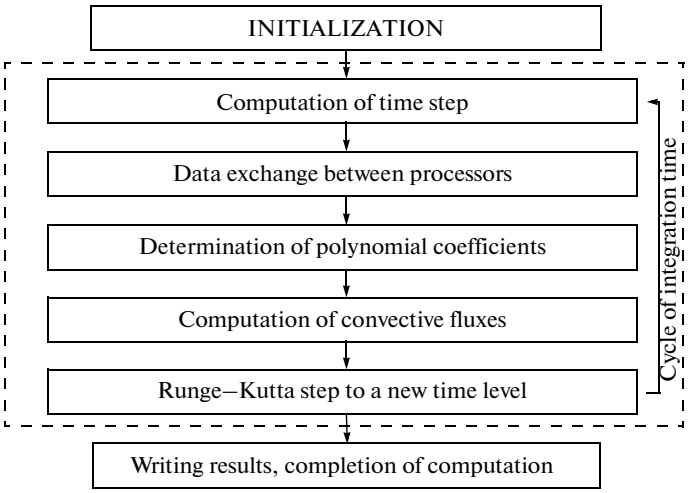

Fig. 2. Block diagram of the computational algorithm.

tion of polynomial coefficients and convective fluxes) are of primary interest. Additionally, the problem is further simplified and convective fluxes are computed only through internal faces of control volumes.

3.1. Determination of Polynomial Coefficients

For a high-order accurate numerical scheme with variables determined at the centers of elements, we use an approach based on polynomial approximation. Before describing the software implementation of the basic computational modules of the algorithm, we define the variables and arrays used to store the parameter values employed in the computations.

Ne is the total number of grid elements.

Nis is the total number of internal faces in mesh cells.

Q is the array of averaged values of gasdynamic variables in grid elements.

AP is the array of averaged geometric coefficients g_{ID} and coordinates of the barycenters of grid elements.

PC is the array of polynomial reconstruction coefficients for gasdynamic variables in grid elements.

STE, *STA* are the arrays containing lists of indices of grid elements forming stencils for determining the polynomial reconstruction coefficients.

MAT is the array of elements of the matrices A_{ID} required for determining the polynomial reconstruction coefficients.

CE is the array of sums of convective fluxes through the faces of mesh cells.

CVF is the array of geometric parameters of internal faces of mesh cells. The structure with face parameters contains the indices of the adjacent cell faces and three components of the area vector.

The data inside the arrays are grouped elementwise; i.e., the averaged value of the nth gasdynamic variable inside the grid element indexed by *ID* is written to the array cell $Q[ID * 5 + n]$. In a similar manner, the sum of the fluxes of the same variable through the faces of this grid element is stored in the cell $CE[ID * 5 + n]$. Inside the array *APC*, the record corresponding to an element consists of nine real numbers, namely, the values of six constant geometric coefficients g_{ID}, followed by three coordinates of the barycenter of the element. The polynomial coefficients within elementwise records in *PC* are grouped according to gasdynamic variables: for each element, five groups (ρ, ρu, ρv, ρw, and E for each variable) consisting of nine coefficients are written in succession.

The number of elements in the stencil of the grid cell indexed by *ID* and the list of indices of these elements are stored in the arrays *STA* and *STE*. The dimension of the stencil in a cell is determined by the difference $NG_{ID} = STE[ID + 1] - STE[ID]$, while the indices of the stencil elements are contained in the cells of the array *STA* starting from $STA[STE[ID]]$ to $STA[STE[ID + 1] - 1]$ inclusive. The coefficients of the $(9 \times NG_{ID})$ matrix A_{ID} are ordered in columns and are stored in the array *MAT* starting from the element $MAT[STE[ID] * 9]$.

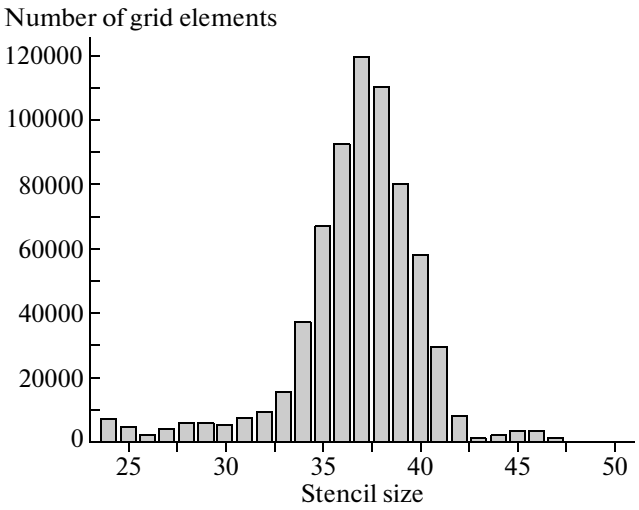

Fig. 3. Histogram of the number of grid elements against the stencil size for the considered test problem (inviscid subsonic flow past a sphere, M = 0.1, grid of 679339 elements).

As was previously said, the procedure for determining polynomial coefficients is rather simple algorithmically and consists of matrix-vector multiplications for each gasdynamic variable in (5). Combining five matrix-vector multiplications for each of the variables yields a single matrix-matrix multiplication: $PC_{ID} = A_{ID} \times B_{ID}$. The elements of the matrix A_{ID} do not vary in the course of the computation. In turn, the columns of the matrix B_{ID} of size $(NG_{ID} \times 5)$ are filled with the differences between the values to five gasdynamic variables in the stencil cells and the cell ID at the current time level.

The iterations of the outer loop over the elements are independent in the sense of writing the results to the array PC. Therefore, to parallelize this loop in a shared memory model, it is sufficient, for example, to use a suitable OpenMP instruction with a static iteration distribution between threads (in fact, between processor cores) specified as an instruction parameter.

However, according to the algorithmic formulation of the problem (see Section 2.2), the stencils of the grid elements can differ in size. Accordingly, the static iteration distribution can be of low performance because of the nonuniform load balancing. Optimal load balancing can be achieved, first, by setting a dynamic iteration distribution as an instruction parameter and, second, by explicitly marking the loop iteration segments to be processed in parallel. In the test numerical experiment considered as an example, the stencil used to determine polynomial coefficients in grid cells is formed by uniting several connection levels of the current cell in the framework of a dual graph. The first connection level consists of the cells sharing a face with the element ID. The second level consists of the elements sharing a face with the elements of the first connection level, etc. The stencil is extended until the prescribed characteristics of the corresponding SLAE are achieved.

Formally, the final difference between the minimal and maximal stencil sizes is rather wide (Fig. 3). However, the actual load unbalance for a static computation distribution is about 10%, and this parallelization variant turns out to be optimal in terms of computation time. Although the multiplication of two matrices is frequently used as an illustration of the benefits of using GPU computations, it is rather problematic to achieve acceptable performance in this case. First, we need to choose the main direction of optimization for the algorithm under development. For this purpose, an elementary task is estimated in terms of the amount of computation per data item. By analogy with the CPU, an elementary task is as yet regarded as determining polynomial coefficients for a grid cell.

In the course of computing polynomial reconstruction coefficients for five gasdynamic variables in the grid cell ID, $116 * NG_{ID} + 408$ bytes of data have to be loaded from and stored in the device memory. The number of floating-point operations is $95 * NG_{ID}$. Thus, for the grid-averaged stencil size $NG_{ID} = 36$, the amount of computation is 0.75 per byte.

In view of the characteristics of AMD 7970 or NVIDIA C2050 devices, for the time required for memory access to be equal to the computation time within a procedure, it is necessary that the number of arithmetic operations per byte be approximately 3.6 (or 29 arithmetic operations per double-precision value). In other words, the amount of computations per data item is considerably less in our case. Therefore, the

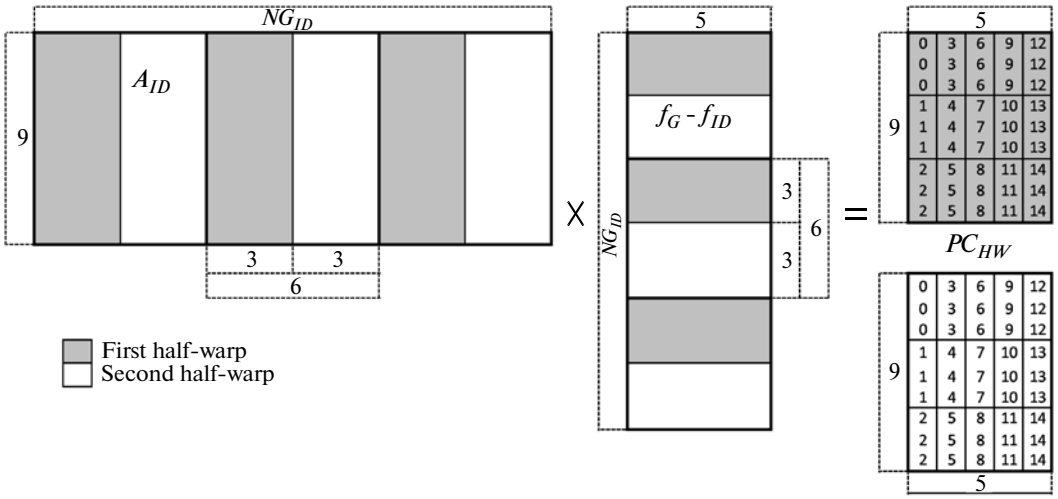

Fig. 4. Block matrix multiplication algorithm. For each position in the blocks of PC_{ID}, the indices of the corresponding active thread in a half-warp are indicated.

performance of the considered subprogram depends nearly completely on the performance of memory access. Consequently, the main direction of algorithmic optimization should be the achievement of the maximum possible coefficient of multiprocessor load. However, in view of the hardware restrictions, this cannot obviously be achieved by treating the processing of a single grid element as an elementary task. It is necessary to find a less resource-intensive unit of parallelism; i.e., the procedure for determining polynomial coefficients in the cell *ID* has to be divided into independent subproblems. The following algorithm was proposed for executing the computations underlying this work. The polynomial reconstruction coefficients are computed by a warp (which is a group of 32 threads executed simultaneously in the framework of SIMD parallelism on a GPU) and this is a cyclic process. The amount of work executed at each step in a cycle consists of the multiplications of six columns of A_{ID} and six rows of B_{ID}. The threads in a warp are divided into two half-warps, since the global memory is simultaneously accessed by a half of warp. The elements of the processed part of A_{ID} are collectively read by the warp threads in shared memory. Next, the threads of the first and second half-warps operate with the corresponding part of the loaded matrix block. The threads of the first half-warp multiply the first three columns of the block by the first three rows of the first half-block of B_{ID}, while the threads of the second half-warp operate with the second half-blocks of A_{ID} and B_{ID}. Thus, at this stage, the desired coefficient matrix PC_{ID} exists in the form of its half-sums for the threads in the first and second half-warps (Fig. 4). Each of the threads of a half-warp, except for the last, is responsible for three consecutive elements in a column of a copy of PC_{ID} corresponding to its half-warp, as denoted in Fig. 4 (the indices in the cells of *PC* blocks correspond to the thread index). In the case of this distribution, the value of only one gasdynamic variable in the cell *ID* is stored in a thread in register memory. The values of the variables corresponding to the column number in a thread for the grid elements within the stencil from the processed part of the block of B_{ID} are also loaded directly in register memory. After the block matrix multiplication procedure is completed, the copies of PC_{ID} obtained in the first and second half-warps are summed in shared memory, followed by writing the final result to the global memory of the device.

The software implementation of the algorithm requires the allocation of 34 registers per thread, which corresponds to 58% load of an NVIDIA C2050 GPU (up to 896 threads can be executed simultaneously). Specifically, for this model, threads are united in blocks of 128 units, and a single block processes four grid cells. It should be noted that, in general, the requirement of the algorithm for the size of the necessary register memory cannot be theoretically estimated. For this reason, the sizes of the multiplied matrix blocks and other parameters affecting the resources required by the algorithm were chosen experimentally in the course of debugging and can vary for various devices.

3.2. Computation of Fluxes through Control Volume Faces

Algorithmically, the procedure used to compute the sum of convective fluxes through internal faces of grid cells is rather simple. If the computation of the flux through a face is regarded as an elementary task, then the following operations are performed for each face in the loop over the faces: the gasdynamic variables are determined on two sides on the face barycenter, the flux through the face is computed (using the Roe scheme), and the resulting values are summed in element stores of the flux array for the adjacent cells.

In the case of a multicore CPU, the basic problem of parallelizing this procedure is to eliminate the situation where fluxes computed by different parallel processes are simultaneously summed in the same cells of the array *CE*. Parallelization variants for flux computation in a shared memory model were described in detail in [11].

If the above algorithm is implemented on GPUs, the number of arithmetic operations per data item is even fewer than in the case of polynomial coefficient determination. A bottleneck is the flux computation based on the Roe scheme (see [10]). This procedure is highly expensive (up to 40 double-precision variables have to be stored in the course of its operation) and algorithmically cannot be split into independent subproblems. Thus, the optimization of an algorithm represented as a single general loop over faces is unpromising. To enhance the performance of the computations in this case, the entire problem has to be split into stages. For example, the computation of sums of convective fluxes can be represented as three sequential stages:

(i) the computation of gasdynamic variables to the left and right of the barycenter of faces;

(ii) the computation of convective fluxes through cell faces;

(iii) the summation of computed fluxes in cells.

The implementation of this approach requires additional arrays (for gasdynamic variables on grid faces, convective fluxes through faces, and indices of cell-belonging faces). However, stages (i) and (iii), which involve reads and writes of large amounts of data, are well parallelized in the framework of the SIMD model. Moreover, a theoretical estimate shows that the total amount of data read from and written to the global memory of the device does not exceed that required in the implementation of the algorithm on a CPU.

Stage (ii) is executed for the set of cell faces, while stages (i) and (iii), for the set of cells. These stages involve an additional data structure (so-called dual graph of control volume connections), which is similar to the arrays *STE* and *STA* in the case of a polynomial stencil and, for each cell, stores the lists of indices of its faces.

It should be noted that the summation stage is necessary to eliminate conflicting accesses to data, and it can be avoided. For example, the set of cell faces can be divided into subsets that do not contain two faces from a single cell. Such a variant was considered, for example, in [12]. When such a subset is processed by parallel threads, the possibility of conflicts related to data modification in the same cells is ruled out by definition. In this implementation variant, a suitable coloring algorithm is applied to the dual graph of the grid, in which the edges correspond to faces, and the nodes to cells. Naturally, in this case, the set of faces is processed in several steps, whose number cannot be less than the maximum number of faces in the cells. However, this variant is much less efficient, even though it involves no additional summation stage. This is associated with the inefficient memory access pattern, which prevents coalescing when data are read from cells. As a rule, the faces are numbered so that they have close, often successive indices within a single cell (at least, for most faces). This means that neighboring threads usually access the same cell, i.e., the same memory data, which leads to coalescing and reduces the time required for data access. If the set of faces is divided into subsets, this situation cannot occur by definition, since subsets do not contain faces belonging to a single cell. The hopelessness of this approach for the given operation is also supported by the results represented in [12].

Consider in more detail the implementation parameters optimized for an NVIDIA C2050 device. For the computation of variables on a face (stage (i))), a block of 320 threads initializes values for 24 grid cells, i.e., initializes values to the left or right of the faces adjacent to the elements. All the threads of the block participate in reading values (such as grid function values, averaged coefficients, and polynomial coefficients) to shared memory. However, only 128 threads (five threads per cell, which corresponds to four warps with the 16th and 32nd threads being inactive) participate in the computation. This initialization configuration is caused by the limited size of shared memory. The low load coefficient and the emulation of register variables in the global memory in the flux computation based on the Roe scheme are associated with the high costs of the algorithm. However, in the case under study, this cannot be avoided in principle. Each thread in the block computes the flux through a single face. At the flux summation stage, a block of 512 threads computes sums for 96 grid elements (5 threads per cell; the 16th threads in the half-warps are

Table 1. Performance for the computation of polynomial reconstruction coefficients

Device	Time, s	Performance, GFLOPS	Percentage of peak performance	Speedup
1-core CPU	0.7078	3.1	26%	1
6-core CPU	0.1450	15.2	21%	4.88
NVIDIA C2050	0.045	50	9.7%	15.6
AMD 7970	0.014	160	17%	50.6

Table 2. Performance of convective flux computation

Device	Time, s	Performance, GFLOPS	Percentage of peak performance	Speedup
1-core CPU	0.261	2.5	21%	1
6-core CPU	0.059	11	16%	4.42
NVIDIA C2050	0.0168	38	7.4%	15.5
AMD 7970	0.0056	115	12%	46.6

inactive). The cost of this procedure is minimal. As a result, the maximal load performance of the GPU can be achieved.

4. PERFORMANCE OF THE COMPUTATIONS

Only double-precision computations were considered. The following devices were used for the tests:

Intel Xeon X5670 CPU, 6 cores, 2.93 GHz frequency, peak performance of 70 GFLOPs, 32 GB/s memory bandwidth, 95 W power;

NVIDIA Fermi C2050 GPU, 14 multiprocessors, 1.15 GHz frequency, peak performance of 515.3 GFLOPs, 144 GB/s memory bandwidth, 238 W power;

AMD Radeon HD 7970 GPU, 32 multiprocessors, 0.92 GHz frequency, peak performance of 947 GFLOPs, 264 GB/s memory bandwidth, 250 W power.

The NVIDIA and AMD GPUs are 7.4 and 13.5 times superior to the CPU in terms of peak performance, respectively. However, the power consumption of the GPUs is roughly 2.5 times higher. A natural estimating criterion for the performance of the GPUs would be their speedup relative to the CPU in excess of the power consumption.

Parallelization over the CPU cores relies on the OpenMP interface [13] for a shared memory model. The software implementation for the GPUs makes use of OpenCL version 1.1 standard (see [14]).

As a test problem for evaluating performance characteristics, we considered computation on a grid of 679339 elements. More specifically, the inviscid subsonic flow past a sphere ($M = 0.1$) was simulated, although the formulation of the computation was of no importance in this context, since the basic algorithmic operations were considered separately.

Table 1 shows the results obtained for operation of computing polynomial reconstruction coefficients. The effective performance and the percentage of the peak performance for a device are given for a single CPU core, the entire 6-core CPU, and the NVIDIA and AMD GPUs. The speedup presented in the table was calculated with respect to the time for a single CPU core. Let us try to estimate the performance of the proposed implementation in terms the producer-announced characteristics of the GPUs. Relying on the maximum read and write rate and the performance of the computations, we theoretically estimate the minimum time required for the execution of the operation. The total amount of global memory reads and writes is 2.96 GB/s. For NVIDIA 2050, the global memory access rate is 144 GB/s, which gives a theoretical time of 0.0205 s required for data copying, while, for AMD 7970 (264 GB/s), this characteristic is 0.011 s. The total amount of computations is 2.21×1024^3 floating-point operations. In view of the peak performance of the devices, the theoretical computation time is 0.0043 s for NVIDIA C2050 and 0.0023 s for AMD 7990. It can be seen that the computation time is much less than the time spent on memory access. The total theoretical time is 0.0248 s for NVIDIA C2050 and 0.0133 s for AMD 7970, which means performance of about 55% for NVIDIA and 95% for AMD without taking into account computations hidden

Table 3. Time distribution over stages in convective flux computation

Stage	Time, s	
	NVIDIA C2050	AMD 7970
1. Computation of values on a face	0.0093	0.0034
2. Computation of flux through a face	0.0055	0.0014
3. Summation	0.002	0.0008
Total	0.0168	0.0056

Table 4. Comparison of GPU and CPU performances

Operation	Speedup with respect to 6-core CPU	
	NVIDIA C2050	AMD 7970
Computation of polynomial coefficients	3.22	10.3
Computation of fluxes through a face	3.51	10.5
Total	3.32	10.4

by memory accesses. To obtain a lower bound, we consider an ideal case when computations are completely hidden by memory accesses. Then the performance is at least 46% for NVIDIA C2050 and 79% for AMD 7970.

Similarly, the results for the operation of convective flux computation are presented in Table 2. Table 3 gives the time distribution between three stages of the operation.

Finally, Table 4 compares the performances of the 6-core CPU and the GPUs. It can be seen that, in all the cases, the resulting speedup exceeds the power relation, which is approximately 2.5.

A comparison of the AMD 7970 and NVIDIA C2050 performances show that, on average, the former is about 3.1 times superior to the latter for the given operations. Note that the ratios of the peak performances and the bandwidths of these devices are roughly 1.8. This suggests that the performance of AMD 7970 is much higher, which can possibly be explained by the insufficient number of registers per thread in NVIDIA (or by the weaker computational architecture as a whole).

To conclude, it should be noted that a similar implementation for each of the operations was also made on the NVIDIA GPU based on the CUDA platform. A comparison of the performances of the CUDA- and OpenCL-based implementations did not reveal noticeable differences for NVIDIA C2050.

5. CONCLUSIONS

The most complicated and computationally costly operations used in a finite-volume CFD algorithm based on a high-order accurate numerical scheme on unstructured hybrid grids were implemented for architectures of massively parallel accelerators. Based on the optimization approaches described, the effective performance of GPUs achieved up to 17% of their peak performance. These characteristics are close to the theoretical limit for this type of operations, which are substantially limited by the memory bandwidth and the irregular memory access pattern. In view of the peak bandwidth, the achieved performance was about 40–50% for NVIDIA C2050 and more than 80% for AMD 7970.

Further studies will be focused on the optimization of the communication scheme for data exchanges between MPI processes and between the CPU and accelerators due to overlapping computations and data transfer. A specialized infrastructure of heterogeneous computations [15] will be used to achieve automated overlapping exchanges and computations.

ACKNOWLEDGMENTS

The authors are grateful to the Keldysh Institute of Applied Mathematics of the Russian Academy of Sciences for the opportunity to use its K100 supercomputer.

This work was supported by the Russian Foundation for Basic Research (project nos. 12-01-33022, 12-01-00486) and by the Ministry of Education and Science (state contract no. 14.514.12.0003).

REFERENCES

1. A. Monakov, A. Lokmotov, and A. Avetisyan, "Automatically tuning sparse matrix-vector multiplication for GPU architectures, high performance embedded architectures and compilers," Lecture Notes Comput. Sci. **5952**, 111–125 (2010).
2. N. Bell and M. Garland, *Efficient Sparse Matrix-Vector Multiplication on CUDA*, NVIDIA Technical Report (2008), NVR-2008-004.
3. M. Garland, S. Le Grand, J. Nickolls, J. Anderson, et al., "Parallel computing experiences with CUDA," IEEE Micro **28** (4), 13–27 (2008).
4. J. Habich, C. Feichtinger, H. Köstler, G. Hager, et al., "Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results," Comput. Fluids **80**, 276–282 (2013).
5. C. Obrecth, F. Kuznik, B. Tourancheau, and J.-J. Roux, "The TheLMA project: A thermal lattice Boltzmann solver for the GPU," Comput. Fluids **54**, 118–126 (2012).
6. P. R. Rinaldi, E. A. Dari, M. J. Vénere, and A. Clausse, "A lattice-Boltzmann solver for 3D fluid simulation on GPU," Simul. Model. Practice Theory **25**, 163–171 (2012).
7. P. Zaspel and M. Griebel, "Solving incompressible two-phase flows on multi-GPU clusters," Comput. Fluids **80**, 356–364 (2013).
8. E. V. Koromyslov, A. A. Siner, and M. V. Usanin, "Video card computation of model nozzle sound generation," *Fourth All-Russia Conference on Computational Experiment in Aeroacoustics, Svetlogorsk, September 19–22, 2012* (MAKS, Moscow, 2012), p. 99.
9. T. Barth, *Numerical Methods for Conservation Laws on Structured and Unstructured Meshes,* Technical Report, VKI for Fluid Dynamics Lecture Series, 2003-03.
10. P. L. Roe, "Approximate Riemann solvers, parameter vectors and difference schemes," J. Comput. Phys. **43** (21), 357–372 (1981).
11. I. V. Abalakin, P. A. Bakhvalov, A. V. Gorobets, A. P. Duben', et al., "NOISETTE parallel code for large-scale aerodynamic and aeroacoustic computations," Vychisl. Metody Program. **13**, 110–125 (2012).
12. A. V. Gorobets, S. A. Sukov, A. O. Zheleznyakov, P. B. Bogdanov, et al., "Application of GPU in hybrid two-level MPI + OpenMP parallelization on heterogeneous computing systems," Vest. Yuzhno-Ural. Gos. Univ., No. 25 (242), 76–86 (2011).
13. OpenMP Application Program Interface, Version 3.1, July 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf.
14. Khronos OpenCL Working Group, OpenCL Specification, Version: 1.1, 2010, http://www.khronos.org/registry/cl/specs/openc-1.1.pdf.
15. A. Efremov and P. Bogdanov, "OpenCL mathematical software infrastructure for heterogeneous computing," *Parallel CFD: Book of Abstracts, May 21–25, 2012* (Atlanta, US, 2012), Vol. 4, pp. 18–19.

Translated by I. Ruzanova