Vortex Element Method Adaptation for Flow Numerical Simulation Using GPU

Ilia K. Marchevsky Sergey R. Grechkin-Pogrebnyakov
iliamarchevsky@mail.ru, grechkin.pogrebnyakov@gmail.com

Abstract

Vortex Element Method is meshless lagrangian CFD method which allows to solve complicated aerodynamic and aeroelastic problems for incompressible flows. In 2D case it can be used for simulation both viscous and inviscid media. The computational cost of unsteady flow simulation using Vortex Element Method is usually much smaller in comparison with other numerical methods (mainly mesh methods); furthermore, the computational cost of aeroelastic problem solving is nearly the same as for flow simulation around the fixed airfoil.

The aim of the present research is to develop new algorithm of vortex element method for its execution on NVidia graphic processing units (GPU). Performance of modern graphical cards can be many tens of times greater than performance of central processors (CPU). However, due to some specific features of GPU effective algorithms development is non-trivial problem. The significant advantage of vortex element methods is low memory requirement for computations, so 100-150 Mbytes of DRAM on GPU is enough for all necessary data uploading to GPU and there is no need to transmit data from GPU to computer and vice versa. All the steps of the numerical algorithm of vortex element method can be improved for their execution on GPU. All specific operators, which cannot be effectively executed on GPU, should be replaced by equivalent subroutines, adapted for the specific architecture.

The developed numerical algorithm and the corresponding computer program can be executed on computers equipped with arbitrary CUDA-compatible GPU (capability version 2.0 or higher). Results of computations are exactly the same as on 'classical' algorithms which don't require GPU, but time of computation is many times smaller: for flow simulation around thin plate (Blausius boundary layer) or around circular cylinder acceleration is about 100 times for GeForce 970 and Tesla C2050 graphic cards.

The developed algorithm allows to use number of numerical schemes for vortex element method, including the most effective and accurate, both for aerodynamic and aeroelastic problems. So it is possible to expand applications of vortex method and reduce computational cost of simulations significantly.

1 Introduction

Computational Flow Dynamics (CFD) problems, as well as coupled hydroaeroelastic problems are the most difficult from a computational point of view due to the nonlinearity of the governing equations and their numerical analysis features (in particular, in boundary layers there is a significant change of flow parameters on a short distance). The approach to solving the problem of computational fluid mechanics is not always obvious; there is a number of different techniques that can be divided into three classes: Eulerian, Lagrangian and hybrid Eulerian-Lagrangian approaches. When solving hydroelastic problems Lagrangian and hybrid methods are preferred, because the flow region changes at every time step.

There is a very important class of problems of subsonic flows simulation when the influence of the compressibility can be neglected. If the region of non-zero vorticity is relatively small (for example, in simulations of the external flows), meshless Lagrangian vortex methods can be very effective: in 2D problems — the Method of Discrete Vortexes [1, 2] and the Viscous Vortex Domains method [3]; in 3D problems — the Closed Vortex Frameworks method [2, 4], the method of vortex segments [5] and their modifications.

The main ways of computations speeding up in vortex methods are the following: fast approximate methods usage [6] and parallel algorithms, based on MPI-technology usage [7, 8]. The scalability of the algorithms of vortex methods is very good: it is shown in [8] that it is easy to achieve 30-times acceleration on 64-core cluster.

At the same time, the systematic researches in the field of vortex methods implementation on graphic accelerators with CUDA-technology are practically absent; some information can be found in [9, 10]. It also should be noted that in the area of hybrid methods graphic accelerators are being used very successfully [11].

The aim of this research is to develop the numerical algorithm for viscous vortex domain method, adapted to graphics accelerators usage, and to analyse its efficiency on some model problems.

2 Governing equations

The two-dimensional problem of external viscous incompressible flow simulation around the immovable rigid airfoil is considered. The density of the flow is assumed to be constant, its motion is described by the continuity equation and the Navier — Stokes equations:

$$\nabla \cdot \vec{V} = 0, \tag{1}$$

$$\frac{\partial \vec{V}}{\partial t} + (\vec{V} \cdot \nabla)\vec{V} = -\frac{\nabla p}{\rho} + \nu \Delta \vec{V}. \tag{2}$$

Here $\vec{V} = \vec{V}(\vec{r}, t)$ is flow velocity, $p = p(\vec{r}, t)$ — pressure, $\rho = \text{const}$ — density, $\nu = \text{const}$ — viscosity coefficient; ∇ and Δ — Hamilton's and Laplacian operators correspondingly. Initial velocity distribution is assumed to be known; boundary conditions consist of perturbations decay at infinity

$$\vec{V}(\vec{r}, t) \to \vec{V}_\infty, \quad p(\vec{r}, t) \to p_\infty \quad \text{when} \quad |\vec{r}| \to \infty$$

and no-slip condition on the airfoil surface K:

$$\vec{V}(\vec{r}, t) = 0 \quad \text{when} \quad \vec{r} \in K.$$

In practical computations we need to compute unsteady velocity field in the flow as well as pressure distribution (in the flow or on the airfoil surface) or integral values of aerodynamic loads acting on the airfoil — drag and lift aerodynamic forces and aerodynamic moment.

3 Brief description of vortex methods

Vortex methods presuppose the vorticity $\vec{\Omega} = \nabla \times \vec{V}$ to be the primary calculated value, and it should be noted that in 2D problems vector $\vec{\Omega}$ has only one nonzero component. The velocity field can be reconstructed from the known vorticity distribution using the Biot — Savar law

$$\vec{V}(\vec{r},t) = \vec{V}_\infty + \iint\limits_S \frac{\vec{\Omega}(\vec{\xi},t) \times (\vec{r} - \vec{\xi}\,)}{|\vec{r} - \vec{\xi}\,|^2} dS_\xi, \tag{3}$$

where the integral is calculated over the flow domain S. The continuity equation (1) is satisfied automatically.

In order to calculate the pressure, the analogue of Bernoulli and Cauchy — Lagrange integrals [12] are very useful, while for the calculation of the integral values of the hydrodynamic loads very simple expressions can be used, which also can be found in [12].

The equation (2) in terms of the vorticity takes the simple form

$$\left.\frac{D\vec{\Omega}}{Dt}\right|_{\vec{V}} = \nu \Delta \vec{\Omega}, \tag{4}$$

where $\left.\frac{D}{Dt}\right|_{\vec{V}} = \frac{\partial}{\partial t} + (\vec{V} \cdot \nabla)$ is the substantial (material) derivative. In case of inviscid incompressible flow simulation, the equation (4) expresses the Helmholtz law, which means that the vorticity is 'frozen' in the liquid.

In number of engineering applications, particularly in the simulation of flow around airfoils with sharp edges and angle points there is no need in viscosity accounting, because the main characteristics of such flows can be obtained within the frameworks of inviscid flow simulation. This approach is the basis of the known Method of Discrete Vortexes (MDV) [1, 2] which has been developed since the 1960s and is well-proven in some branches of industry.

If the viscosity effect is essential it is possible to calculate the so-called diffusive velocity $\vec{W} = -\nu \frac{\nabla \Omega}{\Omega}$. In Viscous Vortex Domain (VVD) method the effective approach is developed for diffusive velocity computation [3, 12]. The equation (4) can be reduced to the form

$$\left.\frac{D\vec{\Omega}}{Dt}\right|_{\vec{V}+\vec{W}} = 0, \tag{5}$$

where by analogy with the substantial derivative the following notation is used:

$$\left.\frac{D}{Dt}\right|_{\vec{V}+\vec{W}} = \frac{\partial}{\partial t} + \big((\vec{V} + \vec{W}) \cdot \nabla\big).$$

The mechanical sense of the equation (5) is that in the flow region S the existing vorticity is transferred with velocity $\vec{V} + \vec{W}$, while the 'new' vorticity is generated only at the boundary of the flow, i.e. on the airfoil surface.

Vortex methods belong to the so-called 'particle methods' [13], and these particles are vortex elements — elementary vorticity fields — circular vortices with constant small radius ε, which are used for simulation of the vorticity distribution in the flow region. Total number N of vortex elements can be quite large (tens-hundreds of thousands and even millions).

The position of each vortex element is characterized by vector $\vec{r}_i$ and circulation Γ_i, $i = 1, \ldots, N$. The influence of the vortex elements on the velocity in the flow at an arbitrary point $\vec{r}$ can be calculated according to the discrete analogue of the Bio — Savar law (3)

$$\vec{V}(\vec{r}, t) = \vec{V}_\infty + \sum_{i=1}^{N} \frac{\Gamma_i}{2\pi} \frac{\vec{k} \times (\vec{r} - \vec{r}_i)}{\max\{|\vec{r} - \vec{r}_i|^2, \varepsilon^2\}}, \tag{6}$$

where $\vec{k}$ is unit vector, which is orthogonal to the flow region.

The basic idea of vortex methods simulate the flow via solution of a system of ordinary differential equations describing the motion of the vortex elements instead of solution of the Navier — Stokes (2) equations or the equivalent partial differential equation (4) and (5). The above-described mechanical sense of equation (5) suggests that the circulations of the vortex elements remain constant, and they move with velocity $\vec{V} + \vec{W}$:

$$\begin{cases} \dfrac{d\Gamma_i}{dt} = 0, \\ \dfrac{d\vec{r}_i}{dt} = \vec{V}(\vec{r}_i) + \vec{W}(\vec{r}_i), \end{cases} \qquad i = 1, \ldots, N. \tag{7}$$

Here $\vec{V}(\vec{r}_i)$ is flow velocity at the position of i-th vortex element, which is calculated by the formula (6), $\vec{W}(\vec{r}_i)$ is diffusive velocity at this point.

Note that there is the other possible way: vortex element position can be constant while their circulation should be recalculated at every time step to simulate the transport of vorticity. This approach is the basis on a class of 'vortex-in-cell' numerical methods [14].

The boundary condition at infinity in the vortex method is satisfied automatically. The no-slip boundary conditions on the airfoil (or no-through condition in case of inviscid flow) is satisfied by the generation of vorticity layer at every time step, which is also simulated by a set of vortex elements. This vortex layer is attached if the flow is inviscid; that means that the vortex elements (which simulates this layer), remain their position and they are replaced by new ones at the next time step. In viscous flow vortex layer is free, which means that the vortex elements move in the flow and they form the vortex wake near the airfoil and behind it.

In order to determine the intensity of the vortex layer on the airfoil calculate the circulations of the vortex elements different approaches and numerical schemes can be used, but all of them lead to a system of linear algebraic equations, which dimension corresponds to airfoil discretization. The most effective, especially when using

VVD, is the method based on the equality to zero of the average value of tangential velocity on the panels — line segments approximating the airfoil [15]; it allows to obtain much more accurate results in comparison with traditional numerical schemes, normally used in VVD method.

4 Software implementation and the computational complexity of vortex methods estimation

The numerical algorithm of flow simulation using vortex methods consists of some operations at every time step (Fig. 1). Normally the simulation requires several thousands or even tens of thousands of time steps.

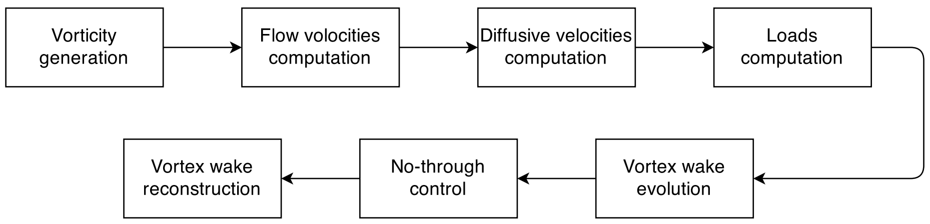
Vorticity generation
Flow volocities computation
Diffusive velocities computation
Loads computation
Vortex wake evolution
No-through control
Vortex wake reconstruction

Figure 1: The scheme of the algorithm

Specific features of the task and the numerical algorithms may affect the implementation of the operations of the shown blocks (in particular, some of the blocks may be absent). We assume that at every time step n vortex elements are generated on the airfoil and total number of vortices in the flow is N ($n \ll N$). In practice n is usually from a few hundred to a few thousand, N can vary from task to task, ranging from several thousand to several hundred thousand. Note that the computation of the velocity generated by a vortex element at an arbitrary point of the flow, as follows from the formula (6), requires 6 arithmetic operations of multiplication/division.

Next, we estimate the complexity of the blocks of the algorithm shown in Fig. 1.

Vorticity generation block. In this block the system of linear algebraic equations is being solved and then circulations of vortex elements generated at the current time step are determined. Matrix of the system is determined only by the shape of the airfoil, so in case of non-deformable airfoil it remains constant. Formation of the matrix and its inversion are carried out only once at the very beginning of the computations, so the complexity of these operations is not taken into account. At every time step only the right side of the linear system is computed, which is based on the calculation of the flow velocity in n points on the airfoil. Linear system solution with the matrix $n \times n$ is reduced to multiplication of the known inverse matrix on the right hand side vector. Thus, the total complexity of vorticity generation block is approximately equal to $(6N + n) \cdot n \approx 6N \cdot n$.

Flow velocities computation block is the most time-consuming. It is necessary to calculate the convective velocities of the vortex elements in the flow. This problem is analogous to N-body problem, its computational complexity with the direct computation of all pairwise influences vortex elements to each other is $6N^2$.

Diffusive velocities computation block. For this block it is harder to estimate the computational complexity because it depends on the specific features of the algorithm and the chosen design scheme. Nevertheless, the diffusive velocities are also depend on vortex pairs interaction and on the effect of the airfoil, so the computational complexity of this block to a first approximation proportional to N^2. Practical results of calculations show that this block is $1{,}5 \dots 2{,}0$ times more difficult than the previous one.

Loads computation block. As mentioned above, various algorithms for hydrodynamic loads calculating can be used depending on the task. In the simplest case, when we calculate only integral loads, which depend only on the pressure distribution on the airfoil, the complexity of this algorithm is extremely low and is proportional to n. Viscous forces accounting requires additional costs, while the complexity becomes proportional to n^2. Calculation of pressure distribution and shear stresses on the airfoil has the complexity proportional to $N \cdot n$.

Vortex wake evolution block provides vortex elements displacement with velocities $\vec{V}_i + \vec{W}_i$, which were found earlier. Thus, its computational complexity is low and proportional to N.

No-through control block is necessary because some vortex elements can penetrate into the airfoil. These vortices should be found and 'extracted' from the airfoil. The computational complexity of this operation, as a first approximation, proportional to n^2.

Vortex wake reconstruction block provides the union (merging) of closely placed vortex elements into one, and the exclusion of the vortex elements, which are far away from the airfoil, because they have practically no influence on the flow parameters near it. The computational complexity of the algorithm, as shown by calculations, proportional to N^2, with the proportionality coefficient is relatively small (less than value 6, that we have for the second block).

5 Model problem

The computer software POLARA [16] allows to simulate viscous incompressible flows around airfoils of arbitrary shape. A parallel implementation of the basic operations of the algorithm can significantly reduce the execution time calculations on multiple-processor computers using MPI-technology.

Consider the Blasius problem of a thin plate simulation in a viscous flow. This problem (for infinitely thin and infinitely long plate) has an exact analytical solution, which can be compared to the results of calculation. The typical form of the wake vortex in the boundary layer on the plate is shown in Fig. 2 [17].

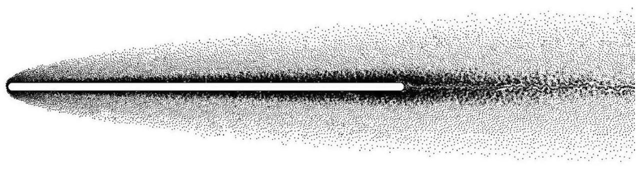

Figure 2: The vortex wake near the plate. Points denote vortex elements positions

The considered plate has a relative thickness of $2\,\%$ and semicircular endings; The Reynolds number based on the length of the plate is 10^3, vortex wake consists of about $120\,000$ vortex elements. The results of the velocity components calculation in the cross section of the boundary layer in comparison with the exact analytical solutions are shown in Fig. 3. Note that the analytical solution is self-similar, so that the position of the cross-section can be selected arbitrarily.

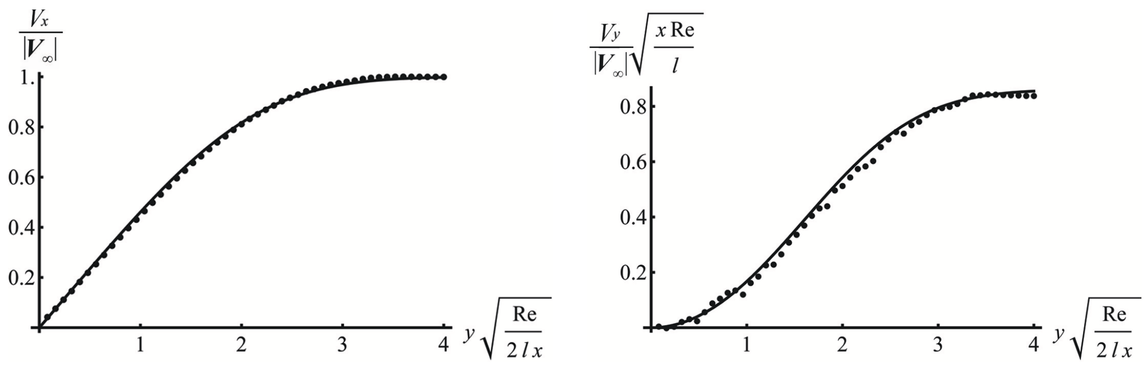

Figure 3: Distribution of the longitudinal (left figure) and transverse (right figure) components of velocity in the boundary layer. The solid line corresponds to the exact analytical solution

There are two main ways to further speed up the calculations. The first is fast approximate methods usage for vortex velocities computation [16]. For this method a very accurate estimation of the computational complexity is obtained in [18], which allows to choose the optimal parameters of the algorithm and reduce the computational complexity from $O(N^2)$ to $O(N \log N)$. The results of numerical experiments [19] show that the acceleration due to fast method usage reaches several tens of times while maintaining acceptable accuracy.

The second approach is graphic cards (GPU) and technology nVidia CUDA usage. These approaches are not mutually exclusive; in relation to the gravitational N-body problem there are effective implementation of approximate methods on GPU [20]. But in the present research we only develop the GPU-adapted algorithm, in which the influences of vortex elements are calculated directly.

6 Features of CUDA-technology usage

An important feature of vortex methods, which allows to use GPUs effectively, is that they require very small memory during the calculation in comparison with mesh methods. Now we estimate the necessary memory (RAM) which is need to run the algorithm of vortex method.

To store the inverse matrix for vortex elements circulations computation we need about 50 MB of memory (in case of 2600 vortex elements approximating the vortex layer on the airfoil). For each vortex element we should keep its circulating, position, velocity vector, as well as the characteristic distance from the nearest to him vortices, therefore, we need $48 \cdot N$ bytes of RAM, where N is number of vortices in the flow. In numerical experiments N does not exceed $330\,000$, so we need for them about 15 MB. When time steps performing ‘old’ and ‘new’ parameters of vortices should be stored in memory, so we need additional 15 MB of memory. In case of multistage Runge — Kutta method usage for the integration of the system (7), which describes the motion of vortices, it will require an additional 15 MB of RAM per each intermediate stage.

Some memory is also required for parameters of the airfoil (not more than a few hundred kilobytes), as well as for overhead data. About 30 MB of GPU memory is reserved by the system when we run any program that uses CUDA technology. Thus, for successful operation of the program in this sample we need approximately 120 MB of RAM while modern graphic cards usually have not less then 1 GB of RAM. Note that these estimates are upper estimates, because values $n = 2\,600$ and $N = 330\,000$ are redundant for most practical purposes, so the real RAM requirement in practice is significantly lower.

Such low RAM requirement simplifies data transfer between the host’s memory and graphic card’s RAM, but its time is still essential, therefore, in the developed algorithm of the vortex method all computations are completely transferred to the GPU, and the data transfer between the GPU and the host only occurs when saving intermediate results. In practice, the saving of the current status of the vortex wake is usually carried out once every few tens or even hundreds of steps; values of the hydrodynamic loads acting on the airfoil are evaluated at every time step, but they are only three real numbers, and their transfer and saving do not cause problems.

Algorithms analysis showed that it should be slightly revisioned for effective execution on GPU. In particular, almost all operations have been implemented without conditional jump if ... else usage. Furthermore, in order to optimize the program all the operation which were previously divided into 7 blocks (Fig. 1), were consolidated into three major blocks, called by the most time-consuming operation: vorticity generation block, velocity computation block and wake reconstruction block.

Velocity computation block provides vortex elements velocity computations, which can be carried out independently. So the approach to parallelization of the operation becomes obvious. Each CUDA-kernel calculates the speed of vortex elements using shared memory, where blocks of N_b vortices are placed in turns.

In **vorticity generation block** each CUDA-kernel calculates the flow velocity at some points on the airfoil. These operations are also independent and they are similar to the computation in the previous block.

Wake reconstruction block includes vortices movement, no-through control and merging of closely placed vortices. For each vortex element several conditions should be checked. Branching operator usage in programs using GPU significantly reduces the speed of the computations, so in order to minimize the negative effects it is necessary to strictly avoid nested branches and to design the algorithm in such a way so that the probability of the first branch is much higher then of the second one. The computational complexity of less probable branch should be many times lower than for the basic branch. Also branchings in the algorithm are not critical in case of extremely low complexities of both paths.

Thus, the operation of vortex elements velocities computation can be effectively parallelized on the GPU, while the efficiency of GPU-parallelization is slightly lower;

a small piece of code (e.g. integral values of hydrodynamic force computation) that has a minimum computational complexity is executed on the GPU in sequential mode.

In this paper, computations were carried out using GPU graphic cards Tesla C2050 and GeForce GTX 970 with 448 and 1644 CUDA-cores respectively and 14 multiprocessors.

The results of numerical experiments show that the vortex methods do not need to perform all computations with double precision. As practice shows, double precision is required only in certain operations inside the vorticity generation and wake reconstruction blocks. This allows to reduce significantly the cost of computations on the GPU, because the performance of graphic cards with single-precision higher (in many cases — several times higher) than when using double precision.

It tables 1 and 2 time is shown which need for 3 000 time steps execution for Blasius problem with single and double precision. Number of vortex elements which are generated on airfoil surface is $n = 2576$, their number in the flow rising from 0 to about 50 000 and its average number is about 43 000.

Table 1: Computational time for GPU Tesla C2050 / GeForce 970, single precision

	Comp. time (seconds) for different block size			
	32	64	128	256
Vorticity generation (B.1)	525/231	538/232	535/228	535/229
Velocity computation (B.2)	3459/1750	2242/1809	2203/1787	2220/1769
Wake reconstruction (B.3)	4026/1871	2573/1610	2365/1610	2455/1659
Total computational time	8010/3852	5353/3651	5103/3625	5210/3657

Table 2: Computational time for GPU Tesla C2050 / GeForce 970, double precision

	Comp. time (seconds) for different block size			
	32	64	128	256
Vorticity generation (B.1)	671/944	678/997	719/995	730/1005
Velocity computation (B.2)	5319/6744	3694/6857	3612/6878	3772/6873
Wake reconstruction (B.3)	3863/1868	2512/1636	2337/1669	2428/1725
Total computational time	9853/9556	6884/9490	6668/9542	6930/9603

Analysis of the data shown in Tables 1 and 2, also shows that in the graphic cards GeForce of the latest generation branch processing implemented more effectively (block B.3), but they are inferior to the relatively old Tesla cards in the performance of a double-precision operations B.1 and B.2 blocks where there are no branching. It also should be noted, that the execution time of the block B.3 is practically the same for single and double precision.

When using GPU Tesla C2050 total execution time with single and double precision varies by 25–30 %, depending on the block size. Numerical experiments show that the optimum block size is 128, changing it to 64 or 128 reduces the productivity of 3–5 %. Too small block size (e.g., 32) leads to 1.5-times increase in time of computations.

Numerical experiments on GPU GeForce GTX 970 show significantly less computational time dependence on the block size, which is caused by the architectural feature of the last GPU generation: all DRAM-memory is cached. However for this GPU single-precision computations (except some operations in B.1 and B.3 blocks mentioned earlier) reduces computational time of B.1 and B.2 blocks in which almost no branching, more than 3.8 times. Execution time of B.3 block remains almost unchanged, so the total time of single precision computations is 2.6 times less than the time for double precision. This feature is extremely important and useful in flow simulating by using vortex methods.

When solving the above described Blasius problem, the following ratios of labor-coefficients have been obtained for basic operations of vortex method (Fig. 4).

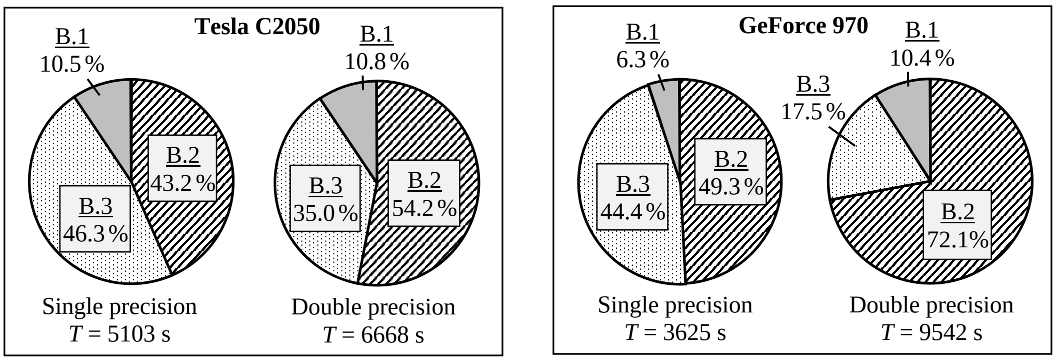

Figure 4: The ratio of labor-coefficients for enlarged blocks in computations on the GPU Tesla C2050 (left figure) and GeForce GTX 970 (right figure) with single and double precision

To assess the overall efficiency of graphic card usage in vortex methods, the same Blasius problem was numerically solved on 'classical' CPU (without GPU usage) in sequential mode (1 core) and in parallel using MPI-technology on a 4-core processor and 16-core cluster (CPU and cluster characteristics can be found in [7]). The time required to perform one time step for $\mathsf{N} \approx 50\,000$ is shown in Table 3.

Table 3: Computational time for 1 time step for $\mathsf{N} \approx 50\,000$

CPU		Cluster, 4 nodes	Tesla C2050		GeForce 970	
1 core	4 cores	CPU, 16 cores	float	double	float	double
211.2 s	57.1 s	17.6 s	2.18 s	2.84 s	1.57 s	4.10 s

As we can see from these results, graphic card Tesla C2050 usage in single-precision mode allows to reduce time of computations more than 8 times in comparison with the 16-core cluster and almost 100 times in comparison with sequential

CPU mode. When computing on the graphic card GeForce GTX 970 (also in single-precision mode), the acceleration is approximately 11 and 135 times, respectively, so execution time for 1 time step is further reduced by a third, despite GeForce GTX 970 has significantly more CUDA-cores compared with Tesla C2050. This effect is explained by the fact that Tesla cards are optimized to perform computations whereas GeForce cards refer to household. Their performance in double-precision mode, as it follows from the results, becomes much lower (more than 2.6 times) and they are noticeably inferior to Tesla cards.

It is also interesting to compare the time of computations in the Blasius problem on GPU using the developed algorithm and on CPU, when the fast method [6] is used in sequential and parallel mode [16, 19]. The accuracy parameter of the fast method was $\theta = 0.2$; in order to choose the optimum depth of the tree the accurate estimation of the complexity for the corresponding algorithm was used [18]. The time required to perform one time step for $N \approx 50\,000$ is shown in Table 4.

Table 4: Computational time for 1 time step for $N \approx 50\,000$ (on CPU — using the fast method)

CPU		Cluster, 4 nodes	Tesla C2050		GeForce 970	
1 core	4 cores	CPU, 16 cores	float	double	float	double
19.69 s	7.92 s	3.44 s	2.18 s	2.84 s	1.57 s	4.10 s

It can be seen from Table 4 that GPU usage provides more than 10-fold acceleration in comparison with sequential calculation on the CPU with fast method (12.5 times for GeForce 970 in single-precision mode), and more than 2-fold acceleration in comparison with the MPI-implementation of the fast method on 16-core cluster. Only double-precision computation on GeForce 970 is 20 % slower than the parallel implementation of the fast method on 16-core cluster.

Conclusion

An nVidia CUDA technolory usage considered as an effective approach for computations acceleration in vortex methods for incompressible flow around airfoils simulation. The algorithm is developed which allows to perform all the calculations on the graphic cards. The estimations for required memory and computational cost of main blocks of the algorithm are obtained. The achieved acceleration values allow to expand significantly vortex methods usage and improve the accuracy of simulating by increasing the number of vortex elements while maintaining a relatively short time of calculation.

The Blasius model problem of flow simulation around thin plate is numerically solved; the obtained results are in good agreement with the exact analytical solutions and known results of numerical experiments.

Acknowledgements

The work was partially supported by Russian Federation President Grant for young scientists [proj. MK-3705.2014.8].

References

[1] Belotserkovskii S.M., Lifanov I.K. Methods of Discrete Vortices. Boca Raton: CRC Press, 1994. 454 p.

[2] Lifanov I.K., Poltavskii L.N., Vainikko G.M. Hypersingular Integral Equations and Their Applications in Mechanics and Physics. Boca Raton: CRC Press, 2003. 408 p.

[3] Dynnikova G.Ya. Vortex Methods for Unsteady Viscous Incompressible Flows Investigation. D.Sc. Thesis (Phys. and Math.). Moscow, 2011. 269 p. (in Russian)

[4] Aparinov V.A., Dvorak A.V. Discrete Vortex Method with Closed Vortex Frameforks // Proceedings of VVIA n.a. N.E. Joukovsky, 1986. Vol. 1313. P. 424–432. (in Russian)

[5] Marchevsky I.K., Shcheglov, G.A. 3D vortex structures dynamics simulation using vortex fragmentons // ECCOMAS-2012, e-Book Full Papers. P. 5716–5735.

[6] Dynnikova G.Ya. Fast technique for solving the N-body problem in flow simulation by vortex methods // Computational Mathematics and Mathematical Physics. 2009. Vol. 49, N 8. P. 1389–1396.

[7] Lukin V.V., Marchevsky I.K., Moreva V.S., Popov A.Yu., Shapovalov K.L., Shcheglog G.A. Computing Cluster for Training and Experiments. Part 2. Examples of Solving Problems // Herald of the Bauman Moscow State Technical University. Series: Natural sciences. 2012. No. 4. P. 82–102. (in Russian)

[8] Marchevsky I.K., Scheglov G.A. Application of parallel algorithms for solving hydrodynamic problems by the vortex element method // Numerical Methods and Programming. 2010. Vol. 11. P. 105–110. (in Russian)

[9] Dynnikova G.Ya., Syrovatsky D.A. 3D Meshless Simulation of Unsteady Fluid Flows by Using the Superhomputers of Hybrid Architecture with Graphic Cards // Proc. of Lomonosov Readings. Moscow, 2012. P. 71–72. (in Russian)

[10] Dynnikova G.Ya., Syrovatsky D.A. Numerical Simulation of 3D Unsteady Incompressible Fluid Flow Around Thin Lifting Surfaces by Using Meshless Method of Dipole Domains // Modern Problems of Aerohydrodynamics: Proc. of XVII school-seminar. Sochi, 2014. P. 55. (in Russian)

[11] Rees W.M., Rossinelli D., Hadjidoukas P., Koumoutsakos P. High performance CPU/GPU multiresolution Poisson solver // Adv. in Parallel Computing. 2014. V. 25. P. 481–490.

[12] Andronov P.R., Guvernyuk S.V., Dynnikova G.Ya. Vortex Methods of Calculation of Unsteady Hydrodynamic Loads. Moscow, MSU Publ., 2006. 184 p.

[13] Li S., Liu W.K. Meshfree Particle Methods. Berlin: Springer-Verlag, 2007. 502 p.

[14] Christiansen J.P. Numerical simulation of Hydrodynamics by the Method of Point Vortices // J. Comp. Phys. 1973. V. 13, No. 3. P. 363–379.

[15] Moreva V.S., Marchevsky I.K. Vortex element method for 2D flow simulation with tangent velocity components on airfoil surface // ECCOMAS 2012 — 6th European Congress on Computational Methods in Applied Sciences and Engineering: Book of proceedings. Vienna, 2012. 14 p.

[16] Moreva V.S., Marchevsky I.K. High-efficiency POLARA program for airfoil aerodynamic characteristics calculation using vortex elements method // The 5-th International Conference on Vortex Flows and Vortex Models: book of proceedings. Caserta (Italy), 2010. P. 1–6.

[17] Makarova M.E., Marchevskii I.K., Moreva V.S. Flow simulation around a thin plate using a modified numerical scheme of the vortex element method // Science and Education. 2013. No. 9. P. 233–242. (in Russian)

[18] Kuz'mina K.S., Marchevskii I.K. Estimation of computational complexity of the fast numerical algorithm for calculating vortex influence in the vortex element method // Science and Education. 2013. No. 10. P. 399–414. (in Russian)

[19] Kuzmina K.S., Marchevskiy I.K. On Computation Speeding Up when Solving Two-Dimensional Hydroelastic Coupled Problems by Using Vortex Methods // PNRPU Aerospace Engineering Bulletin. 2014. No. 39. P. 145–163. (in Russian)

[20] Yokota R., Barba L.A. Hierarchical N-body simulations with auto-tuning for heterogeneous systems // Computing in Science and Engineering. 2012. V. 14. P. 30–39.

Ilia K. Marchevsky, 2-nd Baumanskaya st., 5, Bauman University, Applied Mathematics dep., Moscow, Russia

Sergey R. Grechkin-Pogrebnyakov, 2-nd Baumanskaya st., 5, Bauman University, Applied Mathematics dep., Moscow, Russia