Computers & Fluids 80 (2013) 423–428

Contents lists available at SciVerse ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

Parallel computations on GPU in 3D using the vortex particle method

Andrzej Kosior [*], Henryk Kudela

Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-537 Wroclaw, Poland

ARTICLE INFO

Article history:
Received 21 September 2011
Received in revised form 14 January 2012
Accepted 22 January 2012
Available online 2 February 2012

Keywords:
Vortex in cell
GPU
Multigrid
Parallelization

ABSTRACT

The paper presented the Vortex in Cell (VIC) method for solving the fluid motion equations in 3D and its implementation for parallel computation in multicore architecture of the Graphics Processing Unit (GPU). One of the most important components of the VIC method algorithm is the solution of the Poisson equation. Multigrid and full multigrid methods were chosen for its solution on GPU. Its speed-up was almost 12 times greater than for the direct Fast Poisson Solver for a single processor. The speed-up for the entire VIC method implementation on the GPU was 46 times.

1. Introduction

Many problems in fluid mechanics can be analyzed using vorticity and its evolution in time and space. From this fact there results great importance in those methods that are based on the vortex particle method. In this method particles that carry information about vorticity are used. Tracing the positions of these particles allows one to study how the vorticity has evolved. It is well known that the distribution of vorticity makes it possible to calculate of the fluid velocity.

In the vortex particle method, the particles after several steps have a tendency to concentrate in areas where velocity gradient is very high. It may lead to spurious vortex structures. To avoid this situation after an arbitrary number of time steps, the redistribution of particles to regular positions is done. In 2D we noted [5–7] that it is useful to remesh at every time step. At the beginning the vortex particles are put on numerical mesh nodes. After displacement at every time step the intensities of the particles are redistributed again onto the initial mesh nodes. This strategy has several advantages such as the shortening of computational time and the accurate simulation of viscosity. In the present paper we implemented this idea in 3D flow. Since the computations took very long time, we found that speed-up rendered by parallel computing was necessary.

The VIC method is very well suited for parallel computation. The Poisson equation for each component of the vector potential can be solved independently. The remeshing process, which has to be done at each time step, has a local character and the computations for each particle can be done independently. Also each displacement of the vortex particle has a local character.

To speed-up calculations we decided to use the multicore architecture of the graphics card (GPU). To find out just how much speed-up would be obtained, we decided to conclude some tests. In this paper we show procedures for solving algebraic equations resulting from discretization of the Poisson equation in 3D.

Graphics Processing Units that were developed for video games provide cheap and easily accessible hardware for scientific calculations.

In the next two sections the equations of fluid motion and foundations of 3D VIC method were given. Sections 4 and 5 show how to speed up the calculations using parallel algorithms on multicore architecture. In Section 6 we show a numerical example. In the last section we present a performance of the VIC method using only GPU for calculation.

2. Equations of motion

Equations of incompressible and inviscid fluid motion have the following form:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

where $\mathbf{u} = (u, v, w)$ is velocity vector, ρ is fluid density, p is pressure. The Eq. (1) can be transformed into the Helmholtz vorticity transport equation:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u}\cdot\nabla)\boldsymbol{\omega} = (\boldsymbol{\omega}\cdot\nabla)\mathbf{u} \tag{3}$$

* Corresponding author. Tel.: +48 660462671; fax: +48 713417708.
E-mail addresses: andrzej.kosior@pwr.wroc.pl (A. Kosior), henryk.kudela@pwr.wroc.pl (H. Kudela).

0045-7930/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compfluid.2012.01.014

where $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. From incompressibility (2) stems the existence of vector potential $\mathbf{A}$

$$\mathbf{u} = \nabla \times \mathbf{A} \tag{4}$$

Assuming additionally that vector $\mathbf{A}$ is incompressible ($\nabla \cdot \mathbf{A} = 0$) its components can be obtained by solution of the Poisson equation

$$\Delta A_i = -\omega_i, \qquad i = 1,2,3 \tag{5}$$

Solving (5) one is able to calculate the velocity by formula (4).

In vortex particle methods the viscous splitting algorithm is used. The solution is obtained in two steps: first, the inviscid – Euler equation is solved.

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} \tag{6}$$

In the second step the viscosity effect is simulated by solving the diffusion equation.

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \nu \Delta \boldsymbol{\omega} \tag{7}$$

For the solution of the diffusive equation one can use any suitable method like the Particle Strength Exchange (PSE) method or the Finite Difference method. In the present paper we featured only the solution of 3D equation for inviscid fluid (6).

3. Description of the VIC method for a three-dimensional case

First we have to discretize our computational domain. To do this, we set up a regular 3D numerical mesh $(j_1\Delta x, j_2\Delta y, j_3\Delta z)$ $(j_1, j_2, j_3 = 1, 2, \ldots, N)$, where $\Delta x = \Delta y = \Delta z = h$. The same mesh will be used for solving the Poisson equation. The continuous field of vorticity is replaced by a discrete distribution of the Dirac delta measures [2,8]

$$\boldsymbol{\omega}(\mathbf{x}) = \sum_{p=1}^{N} \boldsymbol{\alpha}_p(\mathbf{x}_p)\delta(\mathbf{x} - \mathbf{x}_p) \tag{8}$$

where $\boldsymbol{\alpha}_p$ means vorticity particle $\boldsymbol{\alpha}_p = (\alpha_{p1}, \alpha_{p2}, \alpha_{p3})$ at position $\mathbf{x}_p = (x_{p1}, x_{p2}, x_{p3})$. The i-th component of the vector particle $\boldsymbol{\alpha}_i$ is defined by the expression:

$$\alpha_i = \int_{V_p} \omega_i(x_1, x_2, x_3)d\mathbf{x} \approx h^3\omega_i(\mathbf{x}_p), \quad \mathbf{x}_p \in V_p, \quad |V_p| = h^3 \tag{9}$$

where V_p is the cell volume with index p.

From the Helmholtz theorems [14] we know that vorticity in inviscid flow is carried out by the fluid.

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p, t) \tag{10}$$

We must also take into account that due to the three-dimensionality of the vorticity field, the intensities of the particles are changed by the stretching effect

$$\frac{d\boldsymbol{\alpha}_p}{dt} = [\nabla \mathbf{u}(\mathbf{x}_p, t)] \cdot \boldsymbol{\alpha}_p \tag{11}$$

The velocity at the numerical mesh nodes was obtained by solving the Poisson Eq. (5) by the finite difference method and (4). In the sequential implementation, this system of equations was solved with the Fast Poisson Solver [11]. (In this work we used an implementation from the FISHPACK numerical library). The system of Eqs. (10), (11) was solved by the Runge–Kutta method of 4th order.

3.1. Remeshing

In the Vortex-in-Cell method, particles have a tendency to gather in regions with high velocity gradients. This can lead to inaccuracies, as the particles are coming too close to one another. To overcome this problem particles have to be remeshed; that is, they have to be distributed back to the nodes of the rectangular mesh. In practice, remeshing is done after several time steps. Usually the simulation of viscosity is done by solving the diffusion equation using the numerical mesh. For this reason it is better to remesh at every time step.

It is done using an interpolation:

$$\omega_j = \sum_p \tilde{\alpha}_{p_n} \varphi\left(\frac{\mathbf{x_j} - \tilde{\mathbf{x}}_p}{h}\right) h^{-3} \tag{12}$$

where j is the index of the numerical mesh node, p is the index of a particle.

Let us assume that $x \in \mathbb{R}$. In this work, we used the following interpolation kernel [2,3,7]

$$\varphi(x) = \begin{cases} (2 - 5x^2 + 3|x|^3)/2 & \text{if } 0 \leqslant |x| \leqslant 1 \\ (2 - |x|)^2(1 - |x|)/2 & \text{if } 1 \leqslant |x| \leqslant 2 \\ 0 & \text{if } 2 \leqslant |x| \end{cases} \tag{13}$$

For the 3D case, $\varphi = \varphi(x)\varphi(y)\varphi(z)$.

We require our interpolation kernel to satisfy

$$\sum_p \varphi\left(\frac{x - x_p}{h}\right) \equiv 1 \tag{14}$$

The discrepancy between the old (distorted) and new (regular) particle distribution, can be measured by the difference

$$\sum_p \tilde{\alpha}_{p_n}\delta(x - \tilde{x}_p) - \sum_p \alpha_{p_n}\delta(x - x_p) \tag{15}$$

In multiplying (15) by a test function ϕ one can get [2]:

$$E = \sum_p \tilde{\alpha}_{p_n}\phi(\tilde{x}_p) - \sum_p \alpha_{p_n}\phi(x_p) \tag{16}$$

where $\tilde{\alpha}_{p_n}$ and $\tilde{x}_p$ are values from old distribution.

Using (12) we can write:

$$E = \sum_p \tilde{\alpha}_{p_n}\left[\phi(\tilde{x}_p) - \sum_j \phi(x_j)\varphi\left(\frac{x_j - \tilde{x}_p}{h}\right)\right] \tag{17}$$

To evaluate error E in (17) we have to evaluate the function

$$f(x) = \phi(x) - \sum_j \phi(x_j)\varphi\left(\frac{x_j - x}{h}\right) \tag{18}$$

Using (14) the Eq. (18) can be rewritten as

$$f(x) = \sum_j [\phi(x) - \phi(x_j)]\varphi\left(\frac{x_j - x}{h}\right) \tag{19}$$

The Taylor expansion of ϕ will yield:

$$f(x) = \sum_j \sum_k \left[\frac{1}{k!}(x_j - x)^k \frac{d^k\phi}{dx^k}\right]^k \varphi\left(\frac{x - x_j}{h}\right) \tag{20}$$

We may conclude, that if φ satisfies the following moment condition [2]:

$$\sum_j (x - x_j)^k \varphi\left(\frac{x - x_j}{h}\right) = 0 \qquad 1 \leqslant |k| \leqslant m - 1 \tag{21}$$

then

$$f(x) = O(h^m) \tag{22}$$

and the remeshing will be of the order m. It means that the polynomial functions up to the order m will be exactly represented by this interpolation.

Kernel (13) used in this work is of order $m = 3$.

4. Parallel computing

As can be seen from the description of the VIC method, most calculations were related to the vortex particles. Those calculations for individual particles are independent from other particles and can be done in parallel. Due to the fact that there are a myriads of vortex particles in the flow (even a few millions) a massively parallel environment is desired. For our calculations we selected Graphics Processing Units (GPUs). They have a great computing power to cost ratio, but required a quite different approach to programming. The GPUs offer a few different types of memory.

One needs to properly recognize problems connected with this architecture in order to take advantage of its computational power. To do that we tested different memory types and numerical algorithms.

GPU's are built according to SIMD architecture in which all computational units execute the same command issued by one instruction unit.

In the following work NVIDIA graphics cards were used. We chose them because it is easy to create programs for them as they use a programming language called "C for CUDA" – an extension of the popular C programming language. The CUDA architecture is built around the multithreaded Streaming Multiprocessors (SMs). Using different graphics cards, we can have different numbers of them. Each SM consists of 8 or 32 Streaming Processors (SPs) (this number depends on the generation of GPU). Code executed on CUDA hardware is divided into threads. A thread is a set of instructions that is executed on a single Streaming Processor. All threads execute the same instructions, but on a different sets of data. To better fit the data, threads are mapped on the grid (one-, two- or three-dimensional). The grid is divided into blocks – groups of threads that will be executed on the same Streaming Multiprocessor. Often there are more blocks than SM's. In that case additional blocks wait until one of the Streaming Multiprocessors terminates the current block. We may talk about a block's lifetime as the time during which it is being executed by SM [9]. Division into blocks allows for greater scalability in executing the software on different types of GPU.

Parallelization of particle displacement and remeshing was straightforward. It was enough to map an individual particle to a single thread. More difficult for implementation was solving the Poisson equation on numerical mesh. The Fast Poisson Solver [11] used on CPU could not be efficiently used on GPU since that it is a sequential algorithm. We therefore had to find another algorithm. One of the best for our purposes was the Multigrid method.

5. Parallel algorithms for solution of Poisson equation

The 3D Poisson equation for the l-component of the vector potential in cartesian coordinates has the form:

$$\frac{\partial^2 A_l}{\partial x^2}+\frac{\partial^2 A_l}{\partial y^2}+\frac{\partial^2 A_l}{\partial z^2}=-\omega_l \tag{23}$$

To simplify the presentation we depicted the algorithm for the two-dimensional version of the Poisson equation. For 2D flows there is only one non-zero component of vector potential $\mathbf{A} = (0,0,\psi)$ and thus the only one Poisson equation $\Delta\psi = -\omega$ had to be solved. A five point stencil is used for discretization of the Poisson equation on a rectangular numerical mesh (ih_x, jh_y). If we assume additionally that the mesh steps in each direction are equal $(h_x = h_y = h)$, we can rewrite Eq. (23) in the form:

$$\psi_{i,j+1}+\psi_{i+1,j}-4\psi_{i,j}+\psi_{i-1,j}+\psi_{i,j-1}=-\omega_{i,j}h^2 \tag{24}$$

where

$$i=0,1,2,\ldots,N_x \qquad j=0,1,2,\ldots,N_y$$

This way we obtained a set of algebraic equations with an unknown vector of stream function $\psi_{i,j}$ at the mesh nodes. This set of equations was solved by the Gauss–Seidel SOR and Multigrid iterative methods.

5.1. Red–Black Gauss–Seidel SOR method

The Gauss–Seidel method is an iterative method for solving the set of linear equations [1,12]. In sequential computing, calculations of unknowns can be done in lexicographical order shown in the Fig. 1. One can take advantage of this fact and rewrite Eq. (24) in the form:

$$\psi_{i,j}^{(k)}=\frac{1}{4}\left(\omega_{i,j}h^2+\psi_{i,j+1}^{(k-1)}+\psi_{i+1,j}^{(k-1)}+\psi_{i-1,j}^{(k)}+\psi_{i,j-1}^{(k)}\right) \tag{25}$$

Unfortunately it cannot be used in parallel computing in this form because we needed all computations to be independent. What we did here is to split our task into two parts. We divided our numerical mesh, like a chessboard, into red and black nodes. It can be seen in the Fig. 2. In the first step of this method we evaluated values only at the black nodes by using the value of ψ at the red nodes. Thanks to this, the computations of the values at the black nodes were independent of each other and can be parallelized. In the second step we do the same with the red nodes by using new values from the black ones. We could now write new equations in the following form:

$$z_{i,j}^{B(k)}=\frac{1}{4}\left(\omega_{i,j}h^2+\psi_{i,j+1}^{R(k-1)}+\psi_{i+1,j}^{R(k-1)}+\psi_{i-1,j}^{R(k-1)}+\psi_{i,j-1}^{R(k-1)}\right) \tag{26}$$

$$z_{i,j}^{R(k)}=\frac{1}{4}\left(\omega_{i,j}h^2+z_{i,j+1}^{B(k)}+z_{i+1,j}^{B(k)}+\psi_{i-1,j}^{B(k)}+\psi_{i,j-1}^{B(k)}\right) \tag{27}$$

For the Red–Black Gauss–Seidel method we could write:

$$\psi_{i,j}^{(k)}=z_{i,j}^{(k)} \tag{28}$$

But we generalized this by introducing a relaxation parameter Ω:

$$\psi_{i,j}^{(k)}=\psi_{i,j}^{(k-1)}+\Omega\left(z_{i,j}^{(k)}-\psi_{i,j}^{(k-1)}\right) \tag{29}$$

The iterative process converges more quickly if the error function is smoother. Choosing properly the value of Ω, we can upgrade the smoothing properties of the Red–Black Gauss–Seidel method. This property of the iterative scheme is called a smoothing property and the scheme using it is called a smoother [13].

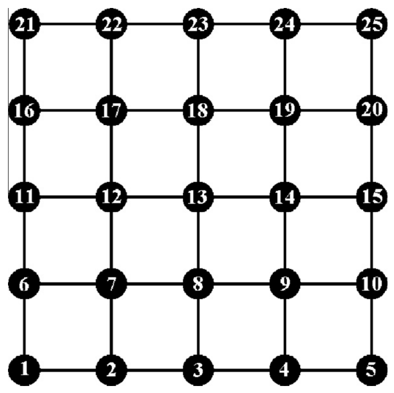

Fig. 1. Lexicographical ordering.

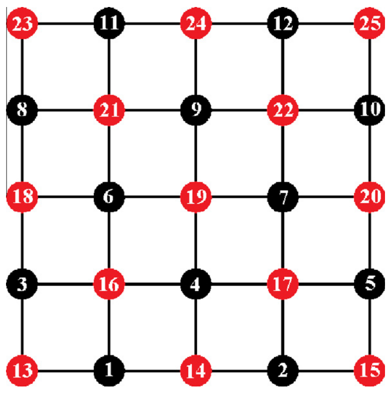

Fig. 2. Red–black ordering (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.).

5.2. Multigrid method

The Gauss–Seidel method is easy to parallelize but cannot compete with the Fast Poisson Solvers. In the multigrid method one uses the fact that the Gauss–Seidel method smoothes the error. The best smoothing relates to the highest frequency error components. A smooth error is well approximated on a coarse mesh. The Gauss–Seidel method on different mesh levels rapidly reduces the corresponding high frequency components and as this process covers all frequencies, a rapid reduction of the overall error can be achieved. It is a very efficient linear solver [13]. Computations can be done in the so-called V-cycle, which can be seen in Fig. 3.

5.3. Full multigrid method

An initial smoothing of the error for iterative solvers like the multigrid, can be achieved by nested iteration. The general idea behind the nested iteration is to provide an initial approximation to the finest mesh from the coarser meshes. It simply means that a lower (coarser) discretization level is used in order to provide a good initial approximation for the iteration on the next higher (finer) level. This combination is called the Full Multigrid (FMG). Typically, the FMG scheme is the most efficient multigrid version [13]. The FMG cycle can be seen in Fig. 3.

6. Test problem

To test the presented algorithms we solved a three-dimensional Poisson equation with a known solution:

$$\psi(x,y,z) = sin(2\pi x) \cdot sin(2\pi y) \cdot sin(2\pi z), x, y, z \in [0,1] \tag{30}$$

with two different boundary conditions: the Dirichlet and the periodic.

We carried out the computation on different meshes and the time of computation we compared with the Fast Poisson Solver executed on a single CPU core.

We tested both the Multigrid and the Full multigrid methods. Computational meshes were prepared so that on each level there

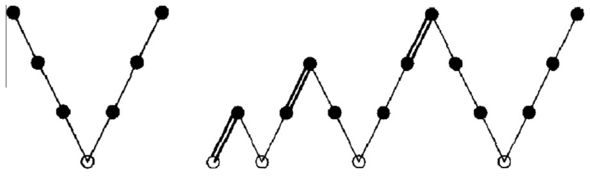

Fig. 3. Different types of computational cycles used in the multigrid method. Left – V-cycle, right – full multigrid cycle.

were $2^n + 1$ mesh points in each direction. (This gives eight levels for the case $257 \times 257 \times 257$). Both methods used the Red–Black Gauss–Seidel smoother and Half-weighting.

The Multigrid method used a simple V-cycle. A different configurations of pre- and post-smoothing steps was tested and finally the V (3,4) configuration was chosen. A stop criteria for the multigrid method was that the maximal residual was lower then $1.0E - 8$.

The Full Multigrid method used eight pre-smoothing and 0 post-smoothing steps.

The final error of calculations in Tables 1 and 2 and speedup is presented in Figs. 4 and 5. Computations were performed on: CPU (Intel Core i7 960), and GPU (NVIDIA GTX 480).

Abbreviations used in Tables 1 and 2 and Figs. 4 and 5: FES – fast elliptic solver, MGD – multigrid method, FMG – full multigrid method.

7. VIC method on GPU

After constructing the Poisson solver on the GPU, we could concentrate on parallelization of the rest of the code. For the time being it was done in a naively. Every function concerning numerical operations on particles was rewritten in C for CUDA.

This made it possible to execute the whole code on the GPU without transferring data between the CPU and the GPU. The initial data was prepared on the CPU and copied to the GPU's memory. From this point all of the calculations were done on the GPU, and memory transfers occurred only when data was being written to files.

To check the correctness of our code, we carried out a test concerning movement of a vortex ring. The calculations were done on a single processor and on a graphics card.

For a thin cored ring with circulation Γ the Kelvin formula for the vortex ring velocity is [4]:

$$U = \frac{\Gamma}{4\pi R_0}\left[\ln\left(\frac{8R_0}{\epsilon_0}\right) - \frac{1}{4} + O\left(\frac{\epsilon_0}{R_0}\right)\right] \tag{31}$$

where R_0 is the ring radius and ϵ_0 is the core radius ($\epsilon_0/R_0 \ll 1$).

We carried out four computational experiments. The parameters of the vortex ring R_0 and ϵ_0 were constant, $R_0 = 1.5$, $\epsilon_0 = 0.3$, but $\Gamma \in [0.25, 1.00]$ was changed. The computational domain was $2\pi \times 2\pi \times 2\pi$ and a numerical mesh had 129 nodes in each direction. Periodic boundary conditions in all directions was assumed.

The initial condition was given in a form that every node which fulfils equation:

$$\left(\sqrt{x^2 + y^2} - R_0\right)^2 + z^2 < \epsilon_0^2 \tag{32}$$

obtained the initial vorticity.

To advance in time the fourth order Runge–Kutta method was used with time step equaling $\triangle t = 0.01$.

The results for the applications running on CPU (Intel Core i7 960) and GPU (NVIDIA GTX 480) are shown in the Figs. 6 and 7.

As one can see in the Figs. 6 and 7, the results for computations using both CPU and GPU for the same number of numerical mesh nodes are nearly identical. The discrepancy may stem from the fact that the Fast Poisson Solver that we used on CPU solved the alge-

Table 1
Final error of different methods with the Dirichlet boundary condition.

Number of mesh points	FES (CPU)	MGD (GPU)	FMG (GPU)
$65 \times 65 \times 65$	8.0358e−4	8.0358e−4	8.1833e−4
$129 \times 129 \times 129$	2.0082e−4	2.0082e−4	2.0374e−4
$257 \times 257 \times 257$	5.0201e−5	5.0201e−5	5.0757e−5

Table 2
Final error of different methods with the periodic boundary condition.

Number of mesh points	FES (CPU)	MGD (GPU)	FMG (GPU)
65 × 65 × 65	8.0358e−4	8.0358e−4	8.2968e−4
129 × 129 × 129	2.0082e−4	2.0082e−4	2.0546e−4
257 × 257 × 257	5.0201e−5	5.0201e−5	5.0835e−5

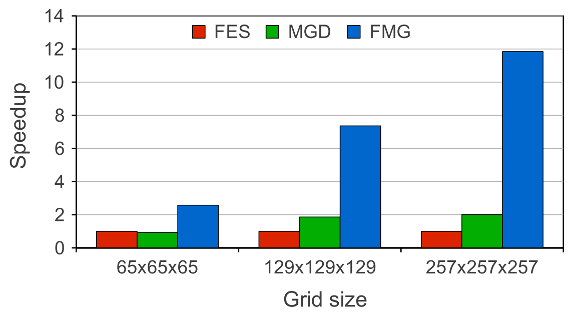

Fig. 4. Speedup of the multigrid method and the full multigrid method comparing to fast elliptic solver with the Dirichlet boundary conditions.

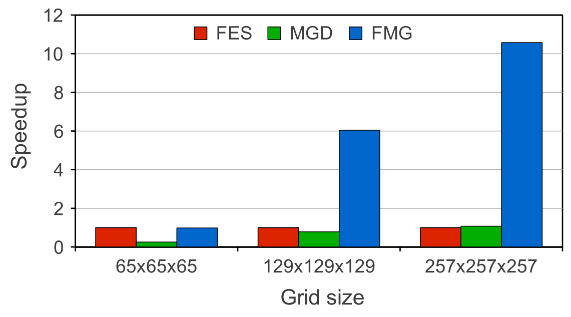

Fig. 5. Speedup of the multigrid method and the full multigrid method compared to the fast elliptic solver with a periodic boundary condition.

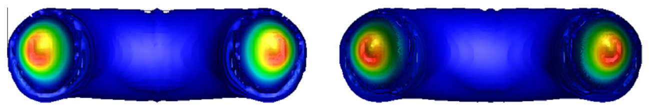

Fig. 6. Results from the VIC method after 600 time steps ($t = 6.0$) for the case $\Gamma = 1.00$ and 129 nodes in each direction running on different hardware. Left – CPU, right – GPU.

braic equations exactly. On GPU we use the iterative Multigrid method. This shows the correctness of our implementation on a graphics card.

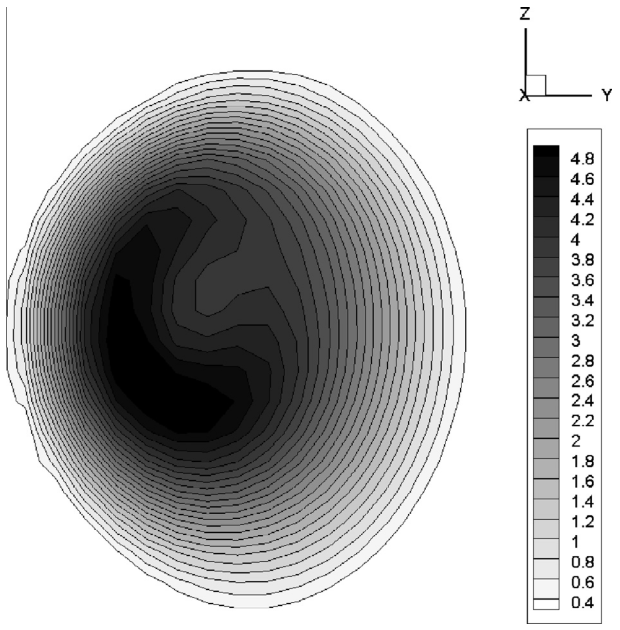

Fig. 8. Isolines of the vorticity in the vortex ring core after $t = 6.0$ for the case $\Gamma = 1.00$ and 129 nodes. Initial vorticity of the core was $|\omega| = 3.54$.

Execution time for the application running on the GPU is 46 times shorter then the one using CPU. This is a very significant speed up.

The discrepancy between the results from the analytical Kelvin formula (31) and our numerical calculation may result from the fact that formula (31) was derived for the vortex ring with the uniform distribution of the vorticity inside the vortex core. During the motion of the ring the vorticity distribution in the core changes in time and is not uniform (see Fig. 8). We noticed also that the increasing number of the grid nodes improves the agreement between the theoretical formula and numerical results. We checked this in computations on a single processor. It is worth noticing that our results better fit to the formula derived by Hicks for stagnant fluids inside the core [10]. In formula of Hicks' the coefficient 1/4 in (31) is replaced by 1/2. In the Fig. 7 the velocity calculated from Hicks formula was drawn using a dashed line.

8. Conclusions

Nowadays it is not difficult to notice that the computational power of a single processor has stopped rising. Parallel architectures need to be used to deliver the means to speed up computa-

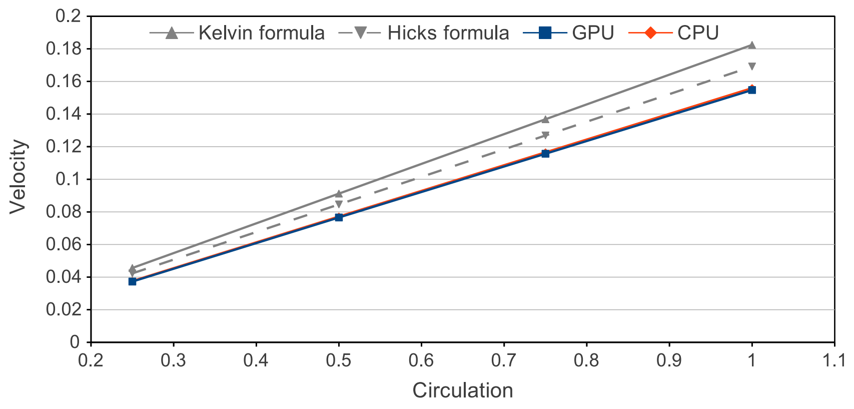

Fig. 7. Velocity of the vortex ring as a function of circulation for calculation on CPU and GPU.

tion. Developing programs on GPU's is an interesting alternative to using the CPU. Thanks to hundreds of streaming processors working in parallel we can get the results faster. The processors also quite cheap and easily accessible.

An important element of parallel computations is choosing the right computational method to allow for the effective use of computer architecture. Moving a sequential program to this hardware may not be the best solution. As we have shown in this article, graphics cards are capable of conducting scientific computations. The discrepancy in the Fig. 7 could result from different methods of solving the Poisson equation. On CPU we used the Fast Elliptic Solver and on GPU we used iterative Multigrid method. As we know Fast Elliptic Solver solves exactly the discretized problem.

It is obvious that if one wants to have a good resolution of the physical phenomena, one has to use a fine numerical mesh in computations. That requires greater memory and computational time. To overcome this problems, one can use many graphics cards. Introducing of multi GPU computation is our aim in the nearest future.

Properly used GPU's (memory management, parallel algorithms, etc.) allows programs to be executed much faster at relatively low cost.

References

[1] Bradie B. A friendly introduction to numerical analysis. Pearson Prentice Hall; 2006.
[2] Cottet GH, Koumoutsakos PD. Vortex methods: theory and practice. Cambridge University Press; 2000.
[3] Cottet GH, Michaux B, Ossia S, VanderLinden G. A comparison of spectral and vortex methods in three-dimensional incompressible flows. J Comput Phys 2002;175:1–11. Academic Press.
[4] Green SI. Fluid vortices. Springer; 1995.
[5] Kudela H, Malecha ZM. Viscous flow modelling using the vortex particles method. Task Quart 2008;13(1–2):15–32.
[6] Kudela H, Malecha ZM. Eruption of a boundary layer induced by a 2D vortex patch. Fluid Dyn Res 2009:41.
[7] Kudela H, Kozlowski T. Vortex in cell method for exterior problems. J Theor Appl Mech 2009;47(4):779–796,.
[8] Kudela H, Regucki P. The vortex-in-cell method for the study of three-dimensional flows by vortex methods. Tubes, sheets and singularities in fluid dynamics, fluid mechanics and its applications, vol. 7. Kluwer Academic Publisher; 2009. p. 49–54.
[9] NVIDIA CUDA C Programming Guide; 2009. <www.nvidia.com>.
[10] Saffman PG. Vortex dynamics. Cambridge University Press; 1992.
[11] Schwarztrauber PN. Fast Poisson solvers. Stud Numer Anal 1984;24:319–70.
[12] Thomas JW. Numerical partial differential equations: conservation laws and elliptic equations. New York: Springer-Verlag; 1999.
[13] Trottenberg U, Oosterlee CW, Schuller A. Multigrid. London: Academic Press; 2001.
[14] Wu JZ, Ma HY, Zhou MD. Vorticity and vortex dynamics. Springer; 2006.