JOURNAL OF THEORETICAL AND APPLIED MECHANICS
50, 1, pp. 285-300, Warsaw 2012
50th anniversary of JTAM

VORTEX PARTICLE METHOD AND PARALLEL COMPUTING

Andrzej Kosior
Henryk Kudela

Wroclaw University of Technology, Institute of Aviation, Processing and Power Machines Engineering, Wroclaw, Poland
e-mail: andrzej.kosior@pwr.wroc.pl; henryk.kudela@pwr.wroc.pl

In this paper, it was presented numerical results related to three dimensional simulation of motion of a vortex ring. For the simulation it was chosen the Vortex In Cell method. The method was shortly described in the paper. The numerical results were obtained on the single processor (x86) architecture. The disadvantage of the single processor computation is a very long time of computation. To menage this problem, we switched to the parallel architecture. In our first approach to the multicore architecture we tested the possibility and algorithms for the solution of the algebraic system of equations that resulted form discretization of the Poisson equation. We presented the results obtained with CUDA architecture. In order to better understand how does the parallel algorithms work on CUDA architecture, it was shortly presented a scheme of the device and how programs are executed on it. We showed also our results which are related to the parallelization of some simple iterative methods like the Jacobi method and Red-Black Gauss-Seidel method for solution of the algebraic system. The results were encouraging. For the Red Black Gauss-Seidel using GTX480 card, the calculations were 90-times shorter than on a single processor. As we know the solution to the Poisson equation is equivalent to the solution to the algebraic systems.

Key words: vortex particle method, vortex ring, parallel computations

1. Introduction

It is well known that vorticity plays a great role in fluid dynamics. Many problems can be analysed using the vorticity and its evolution in time and space. From this fact results a great importance of methods that are based on the vortex particle method. To use them, one introduce particles that carries

information about vorticity. Tracing positions of these particles allows one to study evolution of the vorticity field. From distribution of the vorticity one can calculate fluid velocity. We can distinguish two groups of vortex methods:

- direct methods, where velocity of vortex particles is calculated on the basis of the Biot-Savart law by summing of the contribution to the velocities induced by all particles in the flow,
- Eulerian-Lagrangian methods, like Vortex-In-Cell method (VIC), where the velocity field is determined on a grid by solving the Poisson equation for a vector potential.

The direct methods are very attractive from a theoretical point of view. They are based on the fundamental law of vector analysis, are grid free and allow one to exactly realize the far-field boundary condition. But the direct vortex methods also have a great disadvantage. The amount of computational time is much larger than for the Eulerian-Lagrangian methods (Cottet and Koumoutsakos, 2000). In 2D simulation, the vortex particle method is particularly very efficient. In every time step one must solve only one Poisson equation that combines the component of the vector potential (the stream function) with the third component of vorticity. In 3D computation, in each time step we need to solve three Poisson equations, one for each component of the vector potential.

In the vortex particle method, the particles after several steps have a tendency to concentrate in areas where the velocity gradient is very high. It may lead to spurious vortex structures. To avoid this situation after arbitrary number of time steps the redistribution of particles to regular positions is done. In the 2D case, we noted (Kudela and Malecha, 2008, 2009; Kudela and Kozlowski, 2009) that it is useful to do the remeshing at every time step. At the beginning the vortex particles are put on the grid nodes. After displacement in every time step the intensities of the particles are redistributed again to the initial grid nodes. This strategy has several advantages like shortening of computational time and accurate simulation of the viscosity. In the present paper, we implemented this idea to the case of 3D flow. Due to a very long time of computations we found that the speed-up given by parallel computing is necessary. The VIC method is very good suited for parallel computation. The Poisson equation for each component of the vector potential can be solved independently. The remeshing process has a local character and computations for each particle can be done independently. Also the displacement of the vortex particle can be done in the same manner.

In this work, we present the results of VIC method with the remeshing in each time step applied to 3D motion of the single vortex rings for inviscid

fluid. To speed-up the calculations, we decided to use multicore architectures. It is well suited for solving sets of algebraic equations. To find out how large speed-up can be obtained, we decided to conclude some tests. In this paper, we show procedures solving an algebraic system of equations resulting from discretization of the Poisson equation with the use of multicore architecture. It was found that Graphical Processing Units developed for video games could be used for scientific computations. These Graphical Units provide cheap and easily accessible hardware for scientific calculations. Execution of the sequential algorithm on multicore architecture does not give any speed–up. Parallel computations need special algorithms. In this work, we were using following algorithms for parallel computation:

- Jacobi method,
- Red-Black Gauss-Seidel method.

In the next section the equations of fluid motion are given. In the third section the foundations of 3D VIC method are described. In the fourth section some early tests of the method are shown. In the last two sections it is shown how to speed up the calculations using parallel algorithms and multicore architecture.

2. Equations of motion

Equations of incompressible and inviscid fluid motion have the following form

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\frac{1}{\rho}\nabla p \qquad \nabla \cdot \boldsymbol{u} = 0 \tag{2.1}$$

where $\boldsymbol{u} = [u, v, w]$ is the velocity vector, ρ – fluid density, p – pressure.

Equation $(2.1)_1$ can be transformed to the Helmholtz equation for the vorticity evolution

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} \tag{2.2}$$

where $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$. From incompressibility $(2.1)_2$ stems the existence of the vector potential $\boldsymbol{A}$

$$\boldsymbol{u} = \nabla \times \boldsymbol{A} \tag{2.3}$$

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

$$\Delta A_i = -\omega_i \qquad i = 1, 2, 3 \tag{2.4}$$

3. Description of VIC method for three-dimensional case

First, we have to discretize our computational domain. To do this, we set up a regular 3D grid $(j_1\Delta x, j_2\Delta y, j_3\Delta z)$ $(j_1, j_2, j_3 = 1, 2, \ldots, N)$, 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 (Cottet and Koumoutsakos, 2000; Kudela and Regucki, 2009)

$$\boldsymbol{\omega}(\boldsymbol{x}) = \sum_{p=1}^{N} \boldsymbol{\alpha}_p(\boldsymbol{x}_p)\delta(\boldsymbol{x} - \boldsymbol{x}_p) \tag{3.1}$$

where $\boldsymbol{\alpha}_p$ means the vorticity particle $\boldsymbol{\alpha}_p = (\alpha_{p1}, \alpha_{p2}, \alpha_{p3})$ at the position $\boldsymbol{x}_p = (x_{p1}, x_{p2}, x_{p3})$. The domain of the flow is covered by numerical mesh $(N_x \times N_y \times N_z)$ with equidistant spacing h, and the i-th component of the vector particle $\boldsymbol{\alpha}$ is defined by the expression

$$\alpha_i = \int_{V_p} \omega_i(x_1, x_2, x_3)\, d\boldsymbol{x} \approx h^3\omega_i(\boldsymbol{x}_p) \qquad \boldsymbol{x}_p \in V_p, \qquad |V_p| = h^3 \tag{3.2}$$

From the Helmholtz theorems (Wu *et al.*, 2006) we know that the vorticity is carried on by the fluid

$$\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{u}(\boldsymbol{x}_p, t) \tag{3.3}$$

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

$$\frac{d\boldsymbol{\alpha}_p}{dt} = [\nabla \boldsymbol{u}(\boldsymbol{x}_p, t)] \cdot \boldsymbol{\alpha}_p \tag{3.4}$$

Velocity at the grid nodes was obtained by solving Poisson equation (2.4) by the finite difference method and using (2.3). The system of equations (3.3), (3.4) was solved by the Runge-Kutta method of the 4-th order.

3.1. Remeshing

In the Vortex in Cell method, the particles have a tendency to gather in regions with high velocity gradients. This can lead to inaccuracies as particles come too close to one another. To overcome this problem, the particles have to be remeshed, that is distributed back to the nodes of the rectangular mesh. In practice, the remeshing is done after several time steps. This may cause some difficulty with the accurate simulation of the viscosity that is done by

numerical solution of the diffusion equation. To be able to simulate the viscosity accurately in this work, the particles are remeshed in every time step. It is done using an interpolation

$$\omega_j = \sum_p \widetilde{\Gamma}_{p_n} \varphi\Big(\frac{\boldsymbol{x}_j - \widetilde{\boldsymbol{x}}_p}{h}\Big) h^{-3} \tag{3.5}$$

where j is the index of grid node and p is the index of a particle.

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

$$\varphi(x) = \begin{cases} \dfrac{1}{2}(2 - 5x^2 + 3|x|^3) & \text{if} \quad 0 \leqslant |x| \leqslant 1 \\ \dfrac{1}{2}(2 - |x|)^2(1 - |x|) & \text{if} \quad 1 \leqslant |x| \leqslant 2 \\ 0 & \text{if} \quad 2 \leqslant |x| \end{cases} \tag{3.6}$$

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

We require our interpolation kernel to satisfy

$$\sum_p \varphi\Big(\frac{x - x_p}{h}\Big) \equiv 1 \tag{3.7}$$

To measure the discrepancy between the old (distorted) and new (regular) particle distribution

$$\sum_p \widetilde{\Gamma}_{p_n} \delta(x - \widetilde{x}_p) - \sum_p \Gamma_{p_n} \delta(x - x_p) \tag{3.8}$$

one can multiply the above function by a test function ϕ and get (Cottet and Koumoutsakos, 2000)

$$E = \sum_p \widetilde{\Gamma}_{p_n} \phi(\widetilde{x}_p) - \sum_p \Gamma_{p_n} \phi(x_p) \tag{3.9}$$

where $\widetilde{\Gamma}_{p_n}$ and $\widetilde{x}_p$ are values from the old distribution.

Using (3.5), we can write

$$E = \sum_p \widetilde{\Gamma}_{p_n} \Big[\phi(\widetilde{x}_p) - \sum_p \phi(x_j) \varphi\Big(\frac{x_j - \widetilde{x}_p}{h}\Big)\Big] \tag{3.10}$$

Thus we have to evaluate the function

$$f(\boldsymbol{x}) = \phi(x) - \sum_j \phi(x_j) \varphi\Big(\frac{x_j - x}{h}\Big) \tag{3.11}$$

Using (3.7) the (3.11) can be rewritten as

$$\sum_j [\phi(x) - \phi(x_j)]\varphi\Big(\frac{x_j - x}{h}\Big) \tag{3.12}$$

The Taylor expansion of ϕ yields

$$f(x) = \sum_\alpha \sum_j \Big[\frac{1}{\alpha!}(x_j - x)^\alpha \frac{d^\alpha \varphi}{dx^\alpha}\Big]\varphi\Big(\frac{x - x_j}{h}\Big) \tag{3.13}$$

We get the conclusion that if φ satisfies

$$E = \sum_q (x - x_q)^\alpha \varphi\Big(\frac{x - x_q}{h}\Big) \qquad 1 \leqslant |\alpha| \leqslant m - 1 \tag{3.14}$$

then

$$f(x) = O(h^m) \tag{3.15}$$

and the remeshing procedure 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 (3.6) used in this work is of order $m = 3$.

4. Test problem

To check correctness of our code with the remeshing particle position in every time step we carry out a test for the vortex ring. The calculations were done on a single processor.

For a thin cored ring with circulation Γ formula for the translation velocity of the vortex ring is (Green, 1995)

$$U = \frac{\Gamma}{4\pi R_0}\Big[\ln\Big(\frac{8R_0}{\epsilon_0}\Big) - \frac{1}{4} + O\Big(\frac{\epsilon_0}{R_0}\Big)\Big] \tag{4.1}$$

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, $R_0 = 1.5$, $\epsilon_0 = 0.3$. The computational domain was $2\pi \times 2\pi \times 2\pi$ and a numerical grid had 128 nodes in each direction. Periodic boundary conditions in all directions were assumed.

In the initial step, every vector particle $\boldsymbol{\alpha}_p$ whose position (x_p, y_p, z_p) fulfills the equation

$$(\sqrt{x^2 + y^2} - R_0)^2 + z^2 - \epsilon_0^2 < 0 \tag{4.2}$$

obtains a constant portion of vorticity. The initial circulation of the ring Γ was changed in the range $\Gamma \in [1.06, 4.24]$.

To advance in time, the fourth order Runge-Kutta method was used with the time step equal $\Delta t = 0.01$.

The results are shown in Fig. 1.

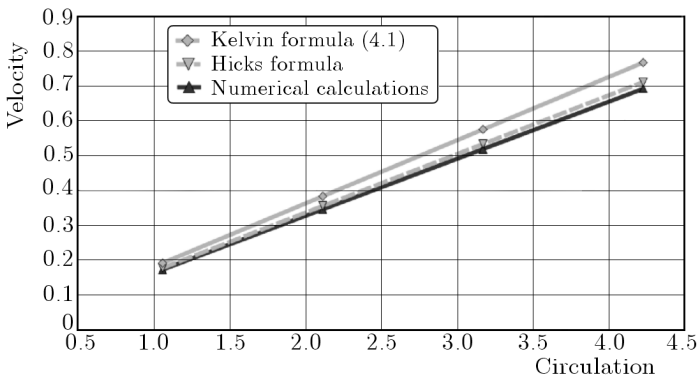

Fig. 1. Comparison of theoretical and computed velocities of the vortex ring

The sequence of positions of the vortex ring for the largest value of Γ is shown in Fig. 2.

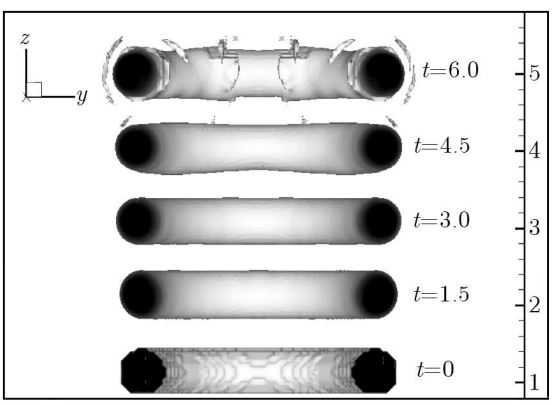

Fig. 2. Vortex ring evolution for the case $\Gamma = 4.23$. The sequence of the iso-vorticity surface for $|\omega| \in [6, 14]$ at different time t

We can see in Fig. 1 good agreement of the theoretical and numerical results. Some discrepancy between the results from analytical Kelvin formula (4.1) and our numerical calculation may stem from the fact that formula (4.1) was derived for the vortex ring with the uniform distribution of the vorticity inside the vortex core. During motion of the ring, the vorticity distribution in the core changes in time and is not uniform (see Fig. 3). It is worth to notice that our results better fit to the formula derived by Hicks for stagnant fluids inside the core (Saffman, 1992). In the formula by Hicks, the coefficient $1/4$ in (4.1) is replaced by $1/2$. In Fig. 1, the velocity calculated from the Hicks formula is drawn by a dashed line.

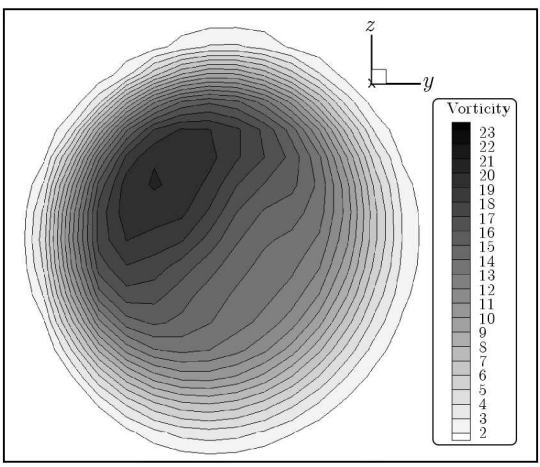

Fig. 3. Isolines of vorticity in the vortex ring core after $t = 6$. The initial vorticity of the core was $|\omega| = 15$

In Fig. 2, we can see the development of the ring in time. The evolution of the vorticity around the core of the ring depend on the parameter of the ring and distribution of the vorticity inside the core. It is well known that the vortex ring is an unstable structure (Green, 1995) and we can expect the ring to take a fuzzy shape. It is clearly visible in Fig. 4 where the position of the vortex ring is shown in the initial and final state but without cutting off the smaller value of the vorticity.

During computation, the invariants of motion were checked. Kinetic energy was calculated from two different equations

$$E_k = \int_{\Omega} \boldsymbol{u}^2 \, dV \qquad \text{and} \qquad E_k = \int_{\Omega} \boldsymbol{A\omega} \, dV \tag{4.3}$$

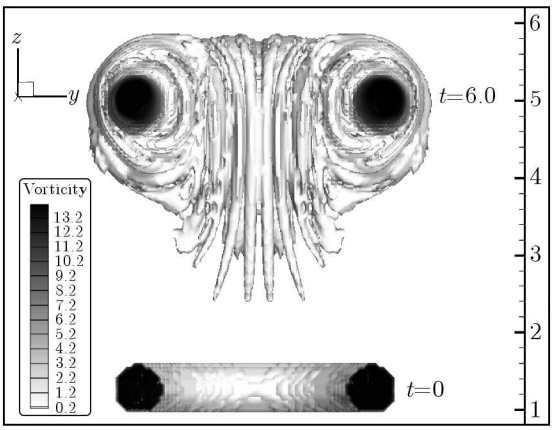

Fig. 4. Vortex ring evolution for the case $\Gamma = 4.23$. The iso-vorticity surface for $|\omega| \in [0.2, 14]$ for the initial and final time step

After 750 time steps, the kinetic energy calculated from equations (4.3) dropped down by less than 2%. The divergences of the vector potential $\boldsymbol{A}$, vorticity $\boldsymbol{\omega}$ and velocity $\boldsymbol{u}$ were all lower than $1.0E-10$ during whole calculations.

5. Parallel computing

The main disadvantage of the presented method is large time of computations. To overcome this problem, it seems to be reasonable to apply parallel computations. The Single Instruction Multiple Data (SIMD) architecture was chosen for parallelization. We may divide our code into two parts:

- part for which the parallelization is straightforward (movement of particles, computation of velocity in grid nodes),
- and part for which the parallelization will cause more problems (solving Poisson equation).

In this paper, we concentrated on the second part of the code, that is on solution of the Poisson equation. First of all, in the three-dimensional case we have to solve three Poisson's equations in every time step. Because they are independent of each other, we can solve them simultaneously. Because Fast Poisson Solvers were designed for sequential computing, they will not take the advantage of SIMD architecture. This is why we need to find some other algorithms.

For our calculations, we have selected Graphical Processing Units (GPUs). It has great computing power-to-cost ratio. But it has a different programming model and offers few types of memory. One need to properly recognize problems connected with this architecture in order to take the advantage of its computational power. To do that, we tested different memory types and numerical algorithms. To find which is the most efficient, we compared them with the same algorithm executed on a single processor. Our results are presented in this paper.

6. Parallel algorithms

Poisson's equation can be written in the following form

$$\frac{\partial^2 A_l}{\partial x^2} + \frac{\partial^2 A_l}{\partial y^2} + \frac{\partial^2 A_l}{\partial z^2} = -\omega_l \tag{6.1}$$

where A_l is a component of the vector potential $\boldsymbol{A} = [A_1, A_2, A_3]$, and ω_l is a component of the vorticity vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]$. The velocity in grid nodes can be calculated from relation (2.3). For two-dimensional flows, there is only one non-zero component of the vector potential $\boldsymbol{A} = [0, 0, \psi]$. For the sake of simplicity, we will derive all of the following formulas for the two-dimensional case. A five-point stencil is used for discretization of the Poisson equation on a rectangular grid (ih_x, jh_y). If we additionally assume that the grid steps in each direction are equal $(h_x = h_y = h)$ we can rewrite equation (6.1) in the form

$$\psi_{i,j+1} + \psi_{i+1,j} - 4\psi_{i,j} + \psi_{i-1,j} + \psi_{i,j-1} = -\omega_{i,j} h^2 \tag{6.2}$$

where

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

This way we obtain a set of algebraic equations with an unknown vector of the stream function $\psi_{i,j}$ at the grid nodes. This set of equations is solved by iterative methods.

6.1. Jacobi method

One of the earliest and simplest methods of solving sets of linear equations is the Jacobi method (Braide, 2006; Thomas, 1999). In every iteration, the values of unknown variables are calculated independently of each other. Thanks to this, the Jacobi method can be easily parallelized. For the two-dimensional

case, in each iteration the value of $\psi_{i,j}$ is computed using $\psi_{i,j}$ values from the previous iteration. One can transform equation (6.2) in order to compute the central element in the form

$$\psi_{i,j}^{(k)} = \frac{1}{4}\Big(\omega_{i,j} + \psi_{i,j+1}^{(k-1)} + \psi_{i+1,j}^{(k-1)} + \psi_{i-1,j}^{(k-1)} + \psi_{i,j-1}^{(k-1)}\Big) \tag{6.3}$$

where k is the iteration number. Because in each iteration the unknown values are independent, one can use parallel computations. Values at each grid node can be evaluated by different threads. Here, the threads are related to the inner structure of the computing device.

6.2. Red-Black Gauss-Seidel method

The second iterative method for solving sets of linear equations is the Gauss-Seidel method (Braide, 2006; Thomas, 1999). In sequential computing, calculations of the unknowns can be done in a lexicographical order shown in Fig. 5. One can see that some values had already been evaluated. For example, to compute the new value for node 13, we use values from nodes 12 and 8 for which the new values were already found. One can take advantage of this fact and rewrite equation (6.3) in the form

$$\psi_{i,j}^{(k)} = \frac{1}{4}\Big(\omega_{i,j} + \psi_{i,j+1}^{(k-1)} + \psi_{i+1,j}^{(k-1)} + \psi_{i-1,j}^{(k)} + \psi_{i,j-1}^{(k)}\Big) \tag{6.4}$$

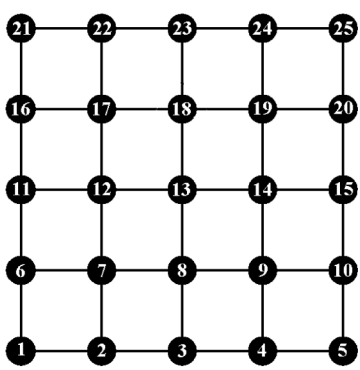

Fig. 5. Lexicographical ordering

Thanks to this, the Gauss-Seidel method needs less iterations than the previous method to solve a set of equations. Unfortunately, it cannot be used in parallel computing in this form because we need all computation to be independent. What we can do here is to split our task into two parts. We divide our computational grid as it is shown in Fig. 6.

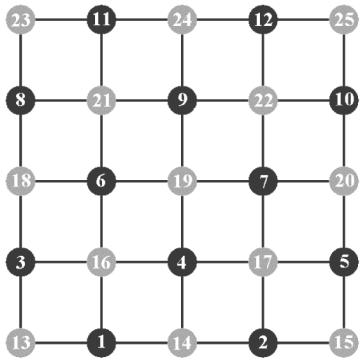

Fig. 6. Red-Black ordering

Like on the chessboard, we divide nodes into red and black ones. In the first step of this method, we evaluate values only at the black nodes. It can be seen that in order to do so we only need values of the unknowns at the red nodes. Thanks to this, the computations will be independent and can be parallelized. In the second step, we do the same with the red nodes using the new values from the black ones. We can write new equations in the following form:

$$\begin{aligned} \psi_{i,j}^{B(k)} &= \frac{1}{4}\Big(\omega_{i,j} + \psi_{i,j+1}^{R(k-1)} + \psi_{i+1,j}^{R(k-1)} + \psi_{i-1,j}^{R(k-1)} + \psi_{i,j-1}^{R(k-1)}\Big) \\ \psi_{i,j}^{R(k)} &= \frac{1}{4}\Big(\omega_{i,j} + \psi_{i,j+1}^{B(k)} + \psi_{i+1,j}^{B(k)} + \psi_{i-1,j}^{B(k)} + \psi_{i,j-1}^{B(k)}\Big) \end{aligned} \tag{6.5}$$

7. Different types of memory in CUDA architecture

An important element of using parallel architectures is efficient usage of the memory. In CUDA architecture, we can distinguish the following memory types [8]:

- device memory
- texture memory
- constant memory
- shared memory
- register memory.

The first of the mentioned types can be accessed by threads from all streaming processors. Unfortunately, reading from this type of memory is very

slow (takes hundreds of clock cycles). During that time no further operations can be done and it slows down execution of the program. If some data is used more than once in the program, there is a way to shorten the access time. We can use shared memory. Access to this memory is very fast (only a few clock cycles) but it can be accessed only by a limited set of threads. Nonetheless, it is useful for solving some problems with memory lags. The making use of it in the test programs enabled a speed-up of over 2 times.

8. Results

8.1. Jacobi method

The test problem was a three-dimensional Poisson equation whose solution was a following function

$$\psi(x,y,z) = 100xyz(x-1)(y-1)(z-1) \qquad x,y,z \in [0,1] \tag{8.1}$$

We tested our method for different types of device memories (shared memory or texture memory). We also tested a case with enlarged computational grid (Fig. 7).

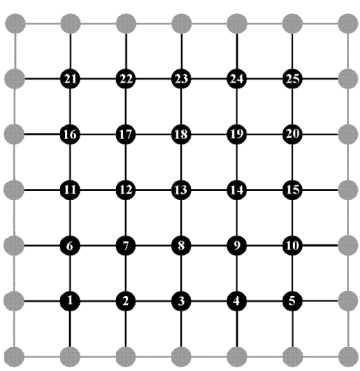

Fig. 7. Enlarged numerical grid allowing for a code with no conditional branching

In this case, one does not have to use conditional branching inside the code for GPU. The parameters of the computational grid and obtained speed-ups are presented in Table 1. In each test, 100 iterations were done by the Jacobi method. Computations were performed on: CPU (Intel Core 2 Quad Q9550), TESLA GPU (NVIDIA TESLA S1070) and FERMI GPU (NVIDIA GFX 480).

Table 1. Speed-up of the Jacobi method

Number of nodes	TSL SH	TSL TX	TSL NC	FRM NC
$32 \times 32 \times 32$	4.05	6.61	6.94	12.32
$64 \times 64 \times 64$	17.71	26.32	31.26	52.82
$128 \times 128 \times 128$	24.78	29.95	43.67	58.89
$256 \times 256 \times 256$	25.47	24.59	41.92	60.96

Abbreviations used in Table 1 and Table 2 denote TSL – TESLA GPU, FRM – FERMI GPU, SH – Shared Memory, TX – texture memory and NC – no conditional branching.

As can be seen, the speed-up strongly depends on both the type of memory used (the results for the global memory which were worse than those for the shared memory are not presented) and the shape of the numerical grid.

8.2. Red-Black Gauss-Seidel

The next test problem was the same equation as for the Jacobi method

$$\psi(x, y, z) = 100xyz(x-1)(y-1)(z-1) \qquad x, y, z \in [0, 1] \tag{8.2}$$

Here we tested only the case which gave the best results for the Jacobi method – the no-conditional branching case. The parameters of the computational grid and obtained results are presented in Table 2. In each test, 100 iterations were done by the Jacobi method. The computations were performed on: CPU (Intel Core 2 Quad Q9550), TESLA GPU (NVIDIA TESLA S1070) and FERMI GPU (NVIDIA GFX 480).

Ttable 2. Speed up of the Red-Black Gauss-Seidel method

Number of nodes	TSL NC	FRM NC
$32 \times 32 \times 32$	11.17	45.06
$64 \times 64 \times 64$	26.11	63.01
$128 \times 128 \times 128$	35.95	88.62
$256 \times 256 \times 256$	31.22	89.47

9. Conclusions

Nowadays, it is not difficult to notice that the computational power of a single processor has stopped rising. Parallel architectures deliver means to speed up computations. Developing programs on GPUs is an interesting alternative

to CPU. Thanks to hundreds of streaming processors working in parallel we can get the results faster. They are also quite cheap and easily accessible. An important element of parallel computations is choosing the right computational method allowing for effective use of the computer architecture. Moving a sequential program to this hardware may not be the best solution. Proper use of GPUs (memory management, parallel algorithms, etc.) allows the programs to be executed much faster (even 60-90 times faster) with a relatively low cost.

Acknowledgments

The authors would like to thank the Institute of Informatics, Wroclaw University of Technology for its support and access to the resources of the Cumulus Computing Environment.

References

1. Braide B., 2006, *A Friendly Introduction to Numerical Analysis*, Pearson Prentice Hall
2. Cottet G.H., Koumoutsakos P.D., 2000, *Vortex Methods: Theory and Practice*, Cambridge University Press
3. Green S.I., 1995, *Fluid Vortices*, Springer
4. Kudela H., Kozlowski T., 2009, Vortex in cell method for exterior problems, *Journal of Theoretical and Applied Mechanics*, **47**, 4, 779-796
5. Kudela H., Malecha Z.M., 2008, Viscous flow modeling using the vortex particles method, *Task Quarterly*, **13**, 1/2, 15-32
6. Kudela H., Malecha Z.M., 2009, Eruption of a boundary layer induced by a 2D vortex patch, *Fluid Dyn. Res.*, **41**
7. Kudela H., Regucki P., 2009, The vortex-in-cell method for the study of three-dimensional flows by vortex methods, [In:] *Tubes, Sheets and Singularities in Fluid Dynamics*, Vol. 7 of *Fluid Mechanics and Its Applications*, 49-54, Kluwer Academic Publisher, Dordrecht
8. *NVIDIA CUDA Programming Guide*, 2009, www.nvidia.com
9. Saffman P.G., 1992, *Vortex Dynamics*, Cambridge University Press
10. Thomas J.W., 1999, *Numerical Partial Differential Equations: Conservation Laws and Elliptic Equations*, Springer-Verlag, New York
11. Wu J.Z., Ma H.Y., Zhou M.D., 2006, *Vorticity and Vortex Dynamics*, Springer

Metoda cząstek wirowych w obliczeniach równoległych

Streszczenie

W pracy przedstawiono wyniki numeryczne ruchu trójwymiarowego pierścienia wirowego. W obliczeniach zastosowano metodę cząstek wirowych, która została pokrótce opisana. Obliczenia przeprowadzono na pojedynczym procesorze (x86). Wadą takiej realizacji jest długi czas obliczeń. Dla przyspieszenia obliczeń zaproponowano algorytm obliczeń równoległych w środowisku wieloprocesorowym karty graficznej z technologią CUDA. Architekturę karty krótko opisano. Znajomość architektury ma istotne znaczenie dla efektywności kodu. Napisany program przetestowano, rozwiązując układ równań algebraicznych otrzymany po dyskretyzacji równania Poissona. Przedstawiono wyniki obliczeń dla zrównoleglonych, prostych metod iteracyjnych rozwiązywania układów równań takich jak metoda Jacobiego czy „Red-Black Gauss-Seidel". Dla metody „Red-Black Gauss-Seidel" oraz karty GTX480 otrzymano 90-krotne przyspieszenie czasu obliczeń względem pojedynczego procesora.

Manuscript received June 25, 2010; accepted for print June 20, 2011