Vortex particle method in parallel computations on graphical processing units used in study of the evolution of vortex structures

2014 Fluid Dyn. Res. 46 061414

(http://iopscience.iop.org/1873-7005/46/6/061414)

IOP Publishing | The Japan Society of Fluid Mechanics Fluid *Dynamics* Research

Fluid Dyn. Res. **46** (2014) 061414 (12pp) doi:10.1088/0169-5983/46/6/061414

Vortex particle method in parallel computations on graphical processing units used in study of the evolution of vortex structures

Henryk Kudela and Andrzej Kosior

Department of Numerical Modelling of Flows, Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland

E-mail: henryk.kudela@pwr.wroc.pl and andrzej.kosior@pwr.wroc.pl

Received 4 July 2013, revised 12 September 2014
Accepted for publication 14 September 2014
Published 16 October 2014

Communicated by Y Fukumoto

Abstract

Understanding the dynamics and the mutual interaction among various types of vortical motions is a key ingredient in clarifying and controlling fluid motion. In the paper several different cases related to vortex tube interactions are presented. Due to problems with very long computation times on the single processor, the vortex-in-cell (VIC) method is implemented on the multicore architecture of a graphics processing unit (GPU). Numerical results of leap-frogging of two vortex rings for inviscid and viscous fluid are presented as test cases for the new multi-GPU implementation of the VIC method. Influence of the Reynolds number on the reconnection process is shown for two examples: antiparallel vortex tubes and orthogonally offset vortex tubes. Our aim is to show the great potential of the VIC method for solutions of three-dimensional flow problems and that the VIC method is very well suited for parallel computation.

(Some figures may appear in colour only in the online journal)

1. Introduction

Numerical solution of the three-dimensional (3D) Navier–Stokes equations for high Reynolds number, using any method, is a very time-consuming process. It may be noted that recently the computational power of a single processor has ceased rising. So in order to increase computing speed we are forced to use multicore architectures and parallel computation.

Graphics processing units (GPUs), developed for video games, provide cheap and easily accessible hardware for scientific calculations. They are built of hundreds of simple streaming processors that altogether provide great computational power. GPUs are relatively cheap and commonly available, but in order to take advantage of their potential one needs to use proper numerical algorithms.

Understanding the dynamics and mutual interaction of various types of vortical motions is the key ingredient in clarifying and controlling fluid motions. One of the most fundamental 3D vortical interactions is related to vortex tube reconnection. Reconnection is regarded as a fundamental mechanism that increases the complexity of the flow which is important in the study of turbulence phenomena (Melander and Hussain 1989).

As a numerical method we chose the 3D vortex-in-cell (VIC) method (Cottet and Koumoutsakos 2000) In this method particles carry information about vorticity. It is well known that the velocity may be calculated from the vorticity distribution. Next, the vortex particles are displaced according to the local velocity field. Particle intensities are then interpolated back to the grid nodes. To simulate the effect of the viscosity, a viscous splitting algorithm was used.

The VIC method is very well suited for parallel computation. The displacement and redistribution processes, which have to be done at each time step, have a local character and the computations for each particle can be done independently. So the whole set of particles can be divided into independent groups, and operations over these groups can be done concurrently.

The structure of the article is as follows: in the next section we give a short description of the VIC method. In section 3 some remarks on parallel computing are given. In section 4 we present the numerical test cases and their results. The last section consists of some closing remarks.

2. Equations of the motion and description of the vortex particle method

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

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu \Delta \boldsymbol{u}, \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u} = (u, v, w)$ is the velocity vector, ρ is the fluid density, p is the pressure, and ν is the kinematic viscosity. Equation (1) can be transformed into the Helmholtz equation for vorticity evolution (Wu *et al* 2006)

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

where $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.

Generally, in all vortex particle methods the viscous splitting algorithm is used (Holden *et al* 2010). The solution is obtained in two steps: First, the inviscid Euler equation is solved:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u}. \tag{4}$$

Next, the viscosity effect is simulated by solving the diffusion equations for each component of vorticity

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \nu \Delta \boldsymbol{\omega}, \tag{5}$$

$$\boldsymbol{\omega}(\boldsymbol{x}, 0) = \boldsymbol{\omega}_I, \tag{6}$$

where $\Delta\boldsymbol{\omega} = (\Delta\omega_1, \Delta\omega_2, \Delta\omega_3)$ and $\boldsymbol{\omega}_I$ is the vorticity distribution obtained after the inviscid step.

For the solution of equations (5) and (6) one can use any suitable method like the particle strength exchange (PSE) method (Cottet and Koumoutsakos 2000) or the finite-difference method. In the present paper we solved (5) and (6) using the implicit finite difference scheme.

In the inviscid flow (4), according to the third Helmholtz theorem (Wu *et al* 2006), vorticity lines move as the material particles of the fluid. Thus, the movement of the vortex particles can be described by the infinite set of ordinary differential equations:

$$\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{u}\left(\boldsymbol{x}_p, t\right), \qquad \boldsymbol{x}(0, \boldsymbol{c}) = \boldsymbol{c}, \tag{7}$$

where $\boldsymbol{c} = (c_1, c_2, c_3)$ means the Lagrange coordinates of fluid particles. Solution of equation (7) gives the particle-trajectory mapping $\Phi(\,\cdot\,, t)$: $\mathbb{R}^3 \to \mathbb{R}^3$; $c \to \Phi(c, t) = \boldsymbol{x} \in \mathbb{R}^3$ that is one-to-one and onto. Incompressibility implies that $\det(\nabla_c \boldsymbol{\Phi}(\boldsymbol{c}, t)) = 1$.

The velocity can be expressed through the vorticity distribution using the Biot–Savart law,

$$\boldsymbol{u}(\boldsymbol{x}, t) = \int \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{x}')\boldsymbol{\omega}(\boldsymbol{x}', t)d\boldsymbol{x}' = (\boldsymbol{K} * \boldsymbol{\omega})(\boldsymbol{x}, t), \tag{8}$$

where * denotes convolution and $\boldsymbol{K}$ is a 3×3 matrix kernel

$$\boldsymbol{K}(\boldsymbol{x}) = \frac{1}{4\pi}\frac{1}{|\boldsymbol{x}|^3}\begin{bmatrix} 0 & x_3 & -x_2 \\ -x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{bmatrix}, \qquad |\boldsymbol{x}| = \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Equation (8) shows the fundamental formulas for direct vorticity methods (Majda and Bertozzi 2002). By covering the domain of the flow with a numerical mesh ($N_x \times N_y \times N_z$) with equidistant spacing h, the ith component of the intensity vector particle $\boldsymbol{\alpha}$ is defined by the expression

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

where V_p is the volume of the cell with index p. The velocity field is found by summing over all of the particles,

$$\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{u}\left(\boldsymbol{x}_p, t\right) = \sum_{k=1, k\neq p}^{N_p} \boldsymbol{K}(\boldsymbol{x}_p - \boldsymbol{x}_k)\boldsymbol{\alpha}_k(t), \tag{10}$$

$$\frac{d\boldsymbol{\alpha}_p}{dt} = \left(\boldsymbol{\alpha}_p(t) \cdot \nabla\right)\boldsymbol{u}\left(\boldsymbol{x}_p, t\right). \tag{11}$$

The disadvantage of the direct vortex particle method given by formula (10) is that it is very computationally time consuming. It stems from the fact that one should take into account all of the mutual interactions between particles that are in the flow. In practical calculation one should also introduce regularization in the formulas (10), removing the singularity of the kernel $\boldsymbol{K}$ (Puckett 1993), (Meiburg 1995), (Majda and Bertozzi 2002). To overcome these difficulties we replaced the Biot–Savart law for velocity calculation with the grid method.

From incompressibility (2) stems the existence of the vector potential $\boldsymbol{A}$, such that

$$\boldsymbol{u} = \nabla \times \boldsymbol{A}. \tag{12}$$

The vector potential $\boldsymbol{A}$ defines the velocity field with accuracy to some potential field $\nabla\phi$. Adding $\boldsymbol{A} + \nabla\phi$, we do not change the left side of (12). It is convenient to assume that $\operatorname{div} \boldsymbol{A} = 0$. The vorticity is related to $\boldsymbol{A}$ as follows:

$$\nabla \times \nabla \times \boldsymbol{A} = \boldsymbol{\omega} = \nabla(\nabla \cdot \boldsymbol{A}) - \Delta \boldsymbol{A}. \tag{13}$$

So, assuming that $\nabla \cdot \boldsymbol{A} = 0$, the vector potential can be obtained by solving three Poisson equations,

$$\Delta A_i = -\omega_i, \qquad i = 1, 2, 3. \tag{14}$$

The Poisson equations can be solved effectively using the numerical grid and finite difference method. The velocity in grid nodes is also obtained by the finite difference using formula (12).

In solving equation (10) by the Runge–Kutta method, we need the velocity of the particles out of the grid nodes. This is obtained using the interpolation of the velocity onto that new particle positions. Using the numerical grid method for solution of equations (10) and (14) significantly accelerates (∼1000 times (Cottet and Poncet 2003)) the calculations. For the solution of the algebraic systems obtained from discretization of Poisson equations (14), we chose a multigrid method, which is very well suited for parallelization. A multigrid method used in a sequential architecture can be as fast as fast Poisson solvers (FPS) (Trottenberg *et al* 2001). The disadvantage of an FPS algorithm that it is a sequential algorithm and cannot take the full advantage of parallel architecture. Our parallel implementation of the multigrid method gave an increased speed of 10 to 12 times that of an FPS algorithm running on a single thread of the processor (Kosior and Kudela 2013). The system of equations (10) was solved by the Runge–Kutta method of the fourth order. In each step, after solution of the equations (10), the obtained intensity of the particles was redistributed onto the grid nodes. This is done using an interpolation,

$$\omega_j = \sum_p \tilde{\alpha}_{p_n} \varphi\left(\frac{\boldsymbol{x}_j - \tilde{\boldsymbol{x}}_p}{h}\right) h^{-3}, \tag{15}$$

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

Let us assume that $x \in \mathbb{R}$. In this article, we use the following interpolation kernel (Cottet and Koumoutsakos 2000):

$$\varphi(x) = \begin{cases} \left(2 - 5x^2 + 3|x|^3\right)/2 & \text{for } 0 \leqslant |x| \leqslant 1, \\ (2 - |x|)^2(1 - |x|)/2 & \text{for } 1 \leqslant |x| \leqslant 2, \\ 0 & \text{for } 2 \leqslant |x|. \end{cases} \tag{16}$$

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

If φ satisfies the moment condition (Cottet and Koumoutsakos 2000)

$$\sum_j (x - x_j)^l \varphi\left(\frac{x - x_j}{h}\right) = 0 \qquad 1 \leqslant |l| \leqslant m - 1, \tag{17}$$

then the redistribution process will be of the order m. This means that the polynomial functions up to the order m will be exactly represented by this interpolation. Kernel (16) used in this work is of order $m = 3$ (Cottet and Koumoutsakos 2000).

3. Remarks on parallel computing on GPUs

As can be seen from the description of the VIC method, most calculations are related to vortex particle displacement and redistribution. These processes are independent of one another and can therefore be done in parallel. For our calculations we selected graphics processing units (GPUs—also referred to as a device in (NVIDIA CUDA C Programming Guide 2012).

Details of implementation of the VIC method on a single GPU can be found in the authors' previous paper (Kosior and Kudela 2013). Introducing the GPU in calculations allowed for a speed-up of about 46 times. This result was obtained on a GTX480 GPU compared to a single thread of an i7 960 @ 3.2 GHz CPU.

A single GPU has its limitations. The most important is the amount of the RAM memory available on a single device. In our vortex particle method implementation, a single GPU used at that time (GTX480 with 1.5GB of RAM) allowed for a numerical mesh not larger than $128 \times 128 \times 128$ nodes. To overcome this limitation we had to write a program capable of using multiple GPUs. This required distribution and exchange of the data between devices. Our final implementation of the VIC method is supposed to run on the computer cluster—i.e., a group of computers (called 'nodes') used as servers connected to each other through a local area network (LAN). Each of the nodes is a standalone computer and can work on its own. Our target hardware configuration is that each node has more than one GPU (in super-computing centres one can find nodes with even up to eight GPUs).

To take advantage of GPUDirect we introduce so-called hybrid MPI-OpenMP parallel programming (also called multilevel parallel programming). OpenMP is a standard for parallel programming on systems with shared memory (in contrast to MPI, which is dedicated to systems with distributed memory). The main difference between these two standards is the fact that in MPI each process makes its own copy of the data that it needs, and in OpenMP processes may read or modify memory spaces created by other processes (variables can be set as a shared or private). The main idea in hybrid MPI-OpenMP programming is that OpenMP ensures communication between the processes running on the same node and MPI transfers the data between the nodes. OpenMP threads can use GPUDirect for the communication between GPUs on the same node.

In order to determine the efficiency of the program using many processes for numerical computations we perform scaling. Informally, scalability concerns the possibility of increasing the acceleration, or equivalently, maintaining a constant efficiency, with the increase in the number of processes used in the calculations. The scalability is affected by the construction of a parallel algorithm, speed of communication between processes and the characteristics of a parallel computer in which the algorithm is executed. There are two types of scaling: strong and weak.

Strong scaling shows the ability to use more processes (GPUs) for the calculations when the computational grid has a constant number of nodes. That may happen if one would like to shorten the computation time of a program. The same total number of nodes are decomposed into more subdomains and distributed among GPUs.

In the weak scaling, which is more important to us, each GPU gets the same number of computational nodes. In the ideal case, when the cards would not need to communicate (or data transfer time would be completely hidden), the computation time for the many-GPUs case would be the same as for a single card. Unfortunately, this is not always possible.

The limiting factor in our test on a single GPU was the amount of RAM memory on a single device, and our main interest was to enlarge the number of grid nodes to raise the precision of the calculations and/or enlarge the computational domain. When using many

devices in computations one needs to ensure the data transfer between them. This takes time and slows down computation. To characterise this loss the weak scaling efficiency is defined as follows:

$$E_w(p, n) = \frac{T(1, n)}{T(p, n)}, \tag{18}$$

where T is the execution time, p is the number of processes and n is the problem size.

Our implementation was tested on a computer with four GTX580 GPUs with 3GB of RAM each. The biggest grid to fit on a single GPU had $224 \times 224 \times 224$ nodes. We tested the weak scaling case. To each GPU a grid of 224^3 nodes is sent. The efficiencies for weak scaling can be seen in table 1.

As one can see, in going from one GPU to two GPUs there is a weak scaling efficiency of 0.91. This is quite a good result. A worse result was obtained using four GPUs. This may be due to the fact that the motherboard used in our computer is capable of transferring data to, at most, two GPUs with the maximum rate given by the PCI Express x16 slot (theoretical peak of 8GB/s and about 6GB/s in tests). When four GPUs are transferring data the transfer is only with half of its maximum rate to each GPU (theoretical peak of 4GB/s and about 3GB/s in tests). This shows that better hardware could improve results.

Current implementation of the VIC method allows for computation on any number of computational nodes having GPUs. Thanks to this we can conduct computations in supercomputing centres as well as on smaller clusters.

4. Numerical verification

To show the correctness of our multi-GPU implementation, two simple test cases are presented: the leapfrogging of two vortex rings in inviscid and viscous fluids, respectively. Leapfrogging strongly depends on initial position of the vortex rings and sometimes may not take place (Lim 1997). During the leapfrogging process, the induced velocity of the front ring will cause the rear ring to contract and accelerate. In contrast, the velocity field of the rear ring will cause the front ring to expand in diameter and slow down. If the conditions are favourable, the rear ring may catch up to the front ring, and be drawn through its centre and emerge ahead of the front ring. When this happens, the role of the rings is reversed and the process may repeat itself.

In both test cases the numerical mesh size is 256^3 and the domain of computation is a box $[20 \times 20 \times 20]$ with periodic boundary conditions in all directions. Distribution of vorticity in the cross section of the ring is assumed as

$$\omega(r) = \omega_0 e^{-\frac{r^2}{a^2}}, \tag{19}$$

where a is the vortex tube radius and ω_0 is the maximal value of vorticity in the tube's cross section at initial time. Based on this, the total circulation Γ and the Reynolds number are calculated as follows:

$$\Gamma = \int_0^a \omega(r) 2\pi r \, dr, \qquad Re_\Gamma = \frac{\Gamma}{\nu}, \tag{20}$$

where ν is the kinematic viscosity coefficient.

Table 1. Weak scaling. Duration of five time steps of the VIC method.

No. of GPUs	Time [ms]	Weak scaling efficiency
1	92090	—
2	101648	0.91
4	159805	0.58

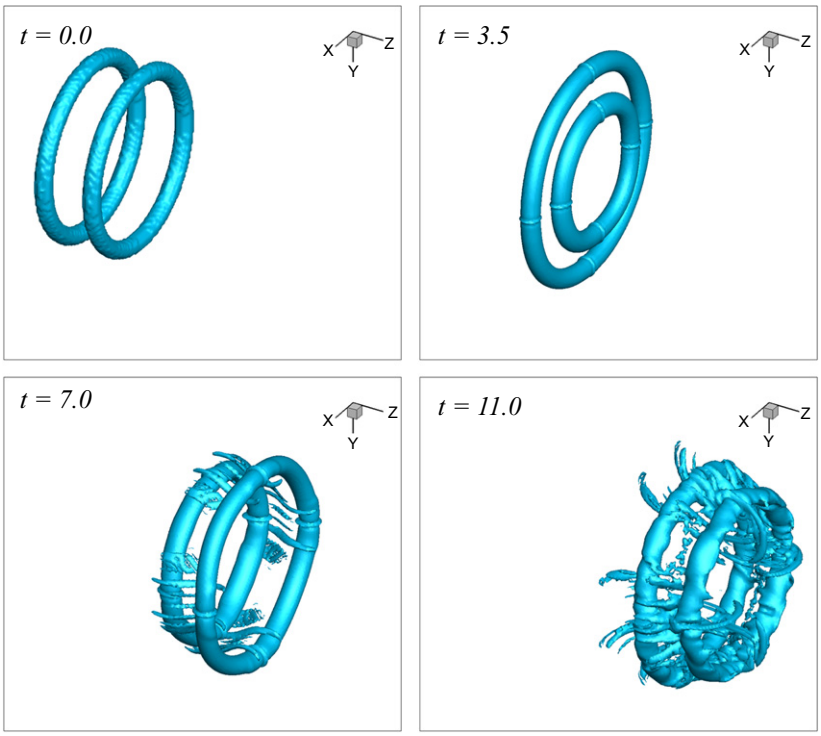

Figure 1. Isosurface plot $|\omega| = 0.2\omega_0$ of leapfrogging in the inviscid fluid.

4.1. Inviscid fluid

The first test is the leapfrogging in an inviscid fluid ($\nu = 0$). The parameters in equations (19) and (20) are $\omega_0 = 22.38$, $a = 0.15$, and $R = 1.5$, $\Gamma = 1.0$.

As one can see in figure 1 the rings undergo two consecutive slip-troughs. Very similar results, for a real-life experiment, can be found in (Lim (1997)).

4.2. Viscous fluid

The second test is conducted for a viscous fluid. The parameters in equations (19) and (20) are the same as for the inviscid case and $Re_\Gamma = 1000$.

In this case vortex rings were able to undergo one slip-through, then merged to form a single ring (figure 2). We think that this is due to the fact that vortex ring cores are diffused during the movement and the interaction between them is too weak to move them apart. Cases 4.1 and 4.2 show that the obtained results are comparable with numerical results obtained by

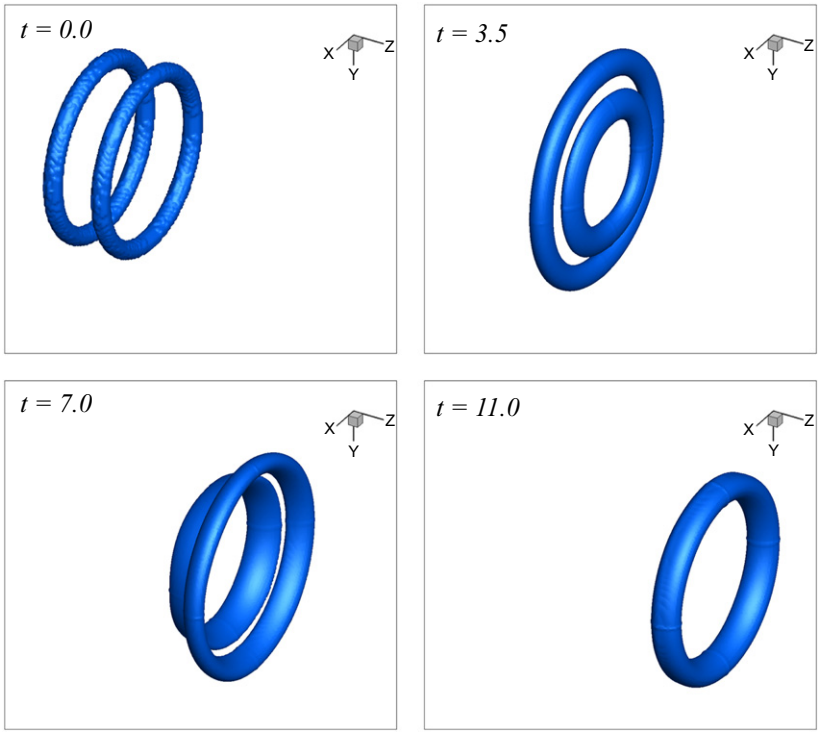

Figure 2. Isosurface plot $|\omega| = 0.15\omega_0$ of leapfrogging in the viscous fluid.

other numerical simulations (Kudela and Regucki 2004) and laboratory experiments (Lim 1997).

5. Influence of the Reynolds number on the reconnection process

Understanding the dynamics and the mutual interaction among various types of vortical motions, including vortex reconnection, is the key ingredient in clarifying and controlling fluid motions (Kida and Takaoka 1994). There is much experimental evidence that tube-like vortex regions evolve and interact at high Reynolds number in 3D turbulent flow. One can imagine that most of the physical space is filled with irrotational or very weakly rotational fluid and that the flow is driven by 'small'-diameter vortex tubes (Zabusky and Melander 1989). The breaking and rejoining of vortex lines may be a fundamental process in the evolution of three-dimensional vortices and the mechanics of turbulence (Melander and Hussain 1989).

The influence of the Reynolds number on the reconnection process will be shown on two examples: reconnection of the identical orthogonally offset tubes and reconnection of the antiparallel tubes.

5.1. Reconnection of the identical orthogonally offset tubes

Values in equations (19) and (20) are $\omega_0 = 20$, $a = 3^{-1/2}$, $\Gamma = 14.73$, $Re_\Gamma = 1403$. The domain of computation is a periodic box $[2\pi \times 2\pi \times 2\pi]$, $\Delta t = 0.001$,

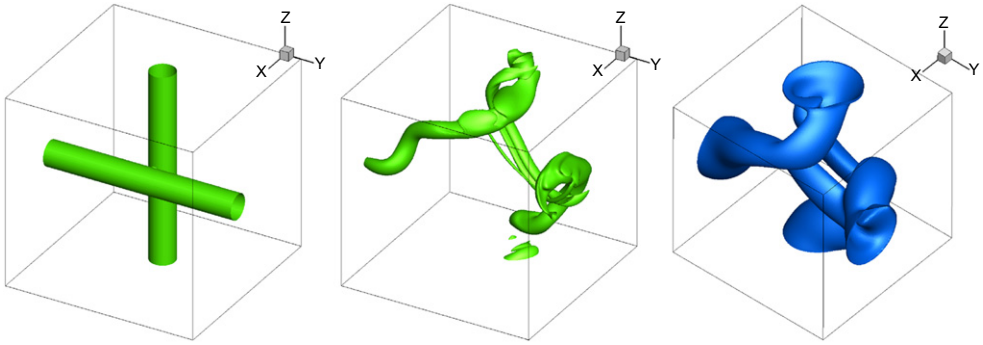

Figure 3. On the left: initial configuration $t = 0.0$; In the middle: $t = 3.0$, $\nu = 0.01$—reconnection takes place; On the right: $t = 3.0$, $\nu = 0.05$—reconnection does not take place.

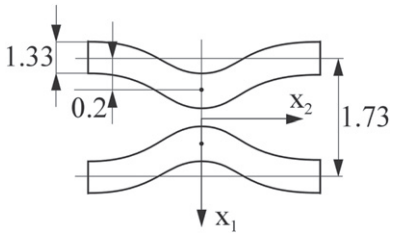

Figure 4. Initial conditions of the antiparallel tubes with symmetric perturbation.

$\Delta x_1 = \Delta x_2 = \Delta x_3 = 2\pi/N$, $N = 256$. The initial data for this test is taken from (Zabusky and Melander 1989) and can be seen on the left-hand side of figure 3. In the middle of figure 3 we can see the evolution of the vorticity field after time $t = 3.0$ with kinematic vorticity coefficient equal to $\nu = 0.01$. As one can see, the reconnection process occurres. On the right-hand side of figure 3 the kinematic viscosity coefficient is raised to $\nu = 0.05$ and there is no reconnection. Only the bridging process (connection between the initial vortex tubes) occurres. Not all vorticity lines are switched, and new separate vortex structures are not created.

5.2. Reconnection of the antiparallel tubes

The geometry of the tubes is given in figure 4. We use a Gaussian vorticity distribution in the core 19.

Initial data is taken from the (Melander and Hussain 1989) and is $a = 0.666$, $\omega_0 = 20.0$, $\Gamma = 17.65$, $Re_\Gamma = 1003$. The computational domain is $[4\pi \times 4\pi \times 4\pi]$. The number of nodes in each directions is $N = 128$ and $\Delta t = 0.01$.

On the left-hand side of figure 5 one can see the reconnection process of the identical orthogonally offset tubes. Quantitative agreement with the results presented in the paper (Melander and Hussain 1989) is very good. On the right-hand side the results for greater viscosity coefficient (lower Reynolds number) are presented. In this case, as before, only the bridging process is observed. There is no reconnection.

Figure 6 shows the time dependence of an absolute maximum $|\omega_{max}|$ with the viscosity ν as a parameter. A similar time-dependence curve (for a single Reynolds number) is reported

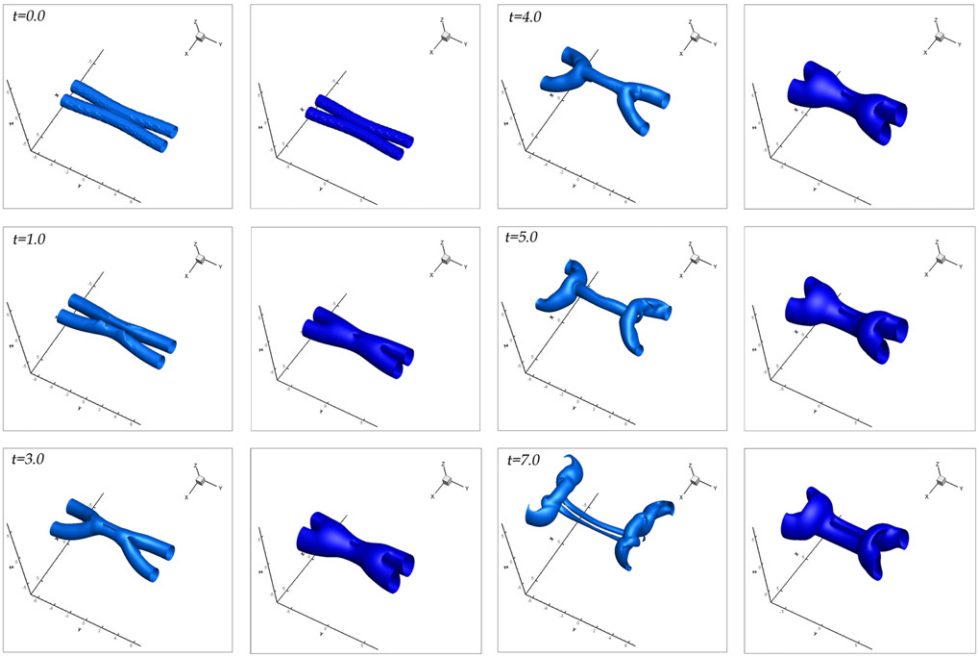

Figure 5. Isosurface plot of the reconnection process of two antiparallel tubes. On the left side $\nu = 0.0176$ (isosurfaces of $0.15\omega_0$), on the right—$\nu = 0.05$ (isosurfaces of $0.05\omega_0$).

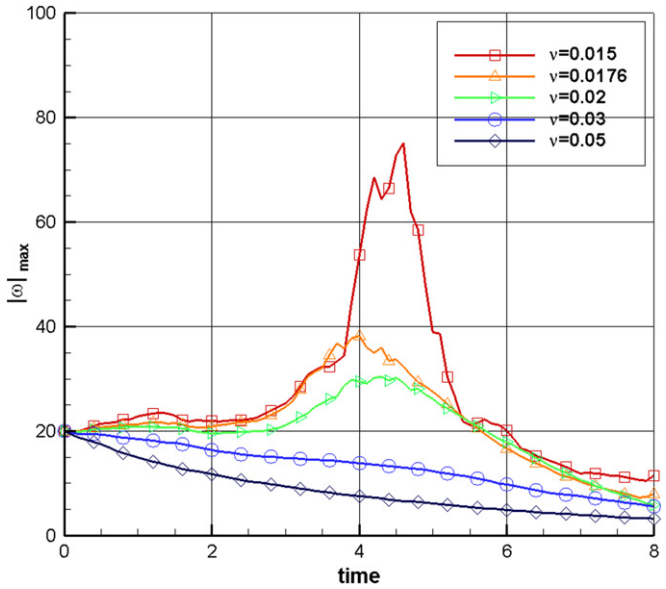

Figure 6. Time dependence of absolute $|\omega_{max}|$ with the viscosity as a parameter.

also in (Kida *et al* 1991). One can notice that with reduction of the value of the viscosity coefficient the slope of the curve grows rapidly and the $|\omega_{max}|$ maximum is greater. This maximum appears at the end of reconnection and start of the threading stage (frame $t = 4$ in figure 5).

It can be clearly seen from figure 6 that peak vorticity occurs only for lower values of the kinematic viscosity coefficient (higher Reynolds numbers). We show in subsections 5.1 and 5.2 that this is closely connected to whether the reconnection process takes place or not. The maximal value of vorticity occurs in the area where the reconnection takes place. To the best of the authors' knowledge this is the first statement of this fact.

6. Closing remarks

We show that our implementation of the vortex particle method is capable of reproducing result obtained in experiments.

Recently, the computational power of a single central processing unit (CPU) has stopped rising. Because of this, parallel architectures need to be used to deliver the means to speed up computation. Developing programs on GPUs is an interesting alternative to using the CPU. Thanks to hundreds of streaming processors working in parallel we can achieve faster results. Such processors are also quite cheap and easily accessible.

It is obvious that if one wants to have a good resolution on physical phenomena, one has to use a fine numerical mesh in computations. This requires greater memory and computational time. To overcome these problems, one can use many GPUs. Introducing multi-GPU computation is our aim in the nearest future. Properly used GPUs (memory management, parallel algorithms, etc) allows programs to be executed much faster at relatively low cost.

We showed that the reconnection process is closely connected to the rise of the maximum value of vorticity in the flow.

References

Cottet G-H and Koumoutsakos P D 2000 *Vortex Methods: Theory and Practice* (Cambridge: Cambridge University Press)

Cottet G-H and Poncet P 2003 Advances in direct numerical simulations of 3D wall-bounded flows by vortex-in-cell methods *J. Comp. Phys.* **193** 136–58

Holden H, Karlsen K H, Lie K-A and Risebro W H 2010 Splitting methods for partial differential equations with rough solutions *EMS Series of Lectures in Mathematics (Zürich)*

Kida S and Takaoka M 1994 Vortex reconnection *Ann. Rev. Fluid Mech.* **26** 169–89

Kida S, Takaoka M and Hussain F 1991 Collision of two vortex rings *J. Fluid Mech.* **230** 583–646

Kosior A and Kudela H 2013 Parallel computations on GPU in 3D using the vortex particle method *Computers & Fluids* **80** 423–8

Kudela H and Regucki P 2004 Vorticity particle method for simulation of 3D flow *Computational Science* vol 3037, ed M Bubak *et al* (Heidelberg: Springer-Verlag) pp 356–63

Lim T T 1997 A note on the leapfrogging between two coaxial vortex rings at low Reynolds numbers *Phys. Fluids* **9** 239–41

Majda A J and Bertozzi A L 2002 *Vorticity and Incompressible Flow* (Cambridge: Cambridge University Press)

Meiburg E 1995 Three-dimensional vortex dynamics simulations *Fluid Vortices* (*Fluid Mechanics and its Applications*) ed S I Green (Dordrecht: Kluwer)

Melander V M and Hussain F 1989 Cross-linking of two antiparallel vortex tubes *Phys. Fluids* A **1** 633–6

NVIDIA 2012 *CUDA C Programming Guide*

Puckett E G 1993 Incompressible computational fluid dynamics trends and advances *Vortex Methods: An Introduction and Survey of Selected Research Topics* ed M D Gunzburger and R A Nicoladies (Cambridge: Cambridge University Press)
Trottenberg U, Oosterlee C W and Schuller A 2001 *Multigrid* (London: Academic)
Wu J Z, Ma H Y and Zhou M D 2006 *Vorticity and Vortex Dynamics* (Berlin: Springer)
Zabusky N J and Melander M V 1989 Three-dimensional vortex tube reconnection: morphology for orthogonally *Offset Tubes Physica* D **37** 555–62