Computers & Fluids xxx (2013) xxx–xxx

Contents lists available at ScienceDirect

Computers & Fluids

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

The 3D vortex particle method in parallel computations on many GPUs

Andrzej Kosior *, Henryk Kudela

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

ARTICLE INFO

Article history:
Received 6 May 2013
Received in revised form 18 September 2013
Accepted 9 October 2013
Available online xxxx

Keywords:
Leapfrogging
Head-on collision
Vortex in cell
Parallel computations
Graphics cards

ABSTRACT

Parallel implementation of the Vortex-in-Cell (VIC) method for 3D flow on many graphics cards was presented. As test problems it was chosen the leapfrogging and head-on collision of two vortex rings for which a well documented visualization exists in the literature. Our aim was to show the great potential of the VIC method for solution of 3D flow problems and that it is very well suited for parallel computation.

1. Introduction

Numerical solution of the 3D Navier – Stokes equations for high Reynolds number, using any method, is a very time consuming process. Recently there is not much increase in the computational power of a single processor. Instead we are forced to use multicore architectures and parallel computation. But to take advantage of their potential one needs to use proper numerical algorithms. One of the multiprocessor hardware that can be efficiently used in scientific computations is the graphics processing unit (GPU). It is built of hundreds of simple streaming processors which altogether give a great computational power. GPUs are relatively cheap and commonly available.

Our first implementation used only one graphics card for computation. We were able to use the computational grid of $128 \times 128 \times 128$ nodes (on a newer GPU we were able to fit computational grid of $224 \times 224 \times 224$ nodes). Our results can be found in [11]. Very quickly we found the limitation of the RAM memory of a single GPU. We were forced to use many graphics cards. For communication we used MPI library.

As a numerical method we chose the 3D Vortex-in-Cell (VIC) method that become more and more important method in numerical investigation of the fluid dynamics phenomena [4–6,20,18]. 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 local velocity field. Particles intensity is then interpolated back to the grid nodes. To simulate the effect of the viscosity the viscous splitting was used and the diffusion equation was solved in each time step.

The VIC method is very well suited for parallel computation [11,10,19,7]. 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.

In the paper it was presented the numerical results of the interaction between two vortex rings. The phenomena of leapfrogging and head-on collision of vortex rings were simulated. Experimental results for both cases are well documented in the literature [12,13].

The structure of the article is as follows: in the next section a short description of the VIC method is given, in Section 3 it was presented the numerical test cases and its results and the last section are 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:

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

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

where $\mathbf{u} = (u, v, w)$ is velocity vector, ρ is fluid density, p is pressure, ν is kinematic viscosity. Eq. (1) can be transformed to the Helmholtz equation for vorticity evolution [21]:

* Corresponding author. Tel.: +48 713203553; 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 © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2013.10.011

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

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

Generally in all vortex particle methods the viscous splitting algorithm is used [8]. The solution is obtained in two steps: first, the inviscid - Euler equation is solved.

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

Next, the viscosity effect is simulated by solving the diffusion equation

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

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

where $\boldsymbol{\omega}_I$ is vorticity distribution obtained after inviscid step.

For the solution of Eqs. (5) and (6) one can use any suitable method like the Particle Strength Exchange (PSE) method [4] or the Finite Difference method (FDM). Due to the fact that we did the distribution of the particles intensities in each time step to the grid nodes and the positions of the particles are fixed, the most suitable method for that case, seems to be the FDM method with semi-implicit Crank–Nicholson scheme that is of the order $(\mathcal{O}((\Delta t)^2 + (\Delta x_i)^2))$.

The mth component ($m = 1,2,3$) of the vorticity vector was approximated by

$$\frac{\omega_m^{n+1} - \omega_m^n}{\Delta t} = \frac{1}{2}L^{n+1}\omega_m + \frac{1}{2}L^n\omega_m \tag{7}$$

where

$$L^{n+1}\omega_m = \nu\left(\Lambda_i\omega_m^{n+1} + \Lambda_j\omega_m^{n+1} + \Lambda_k\omega_m^{n+1}\right) \tag{8}$$

and

$$\Lambda_i\omega = \frac{\omega_{i+1,j,k} - 2\omega_{i,j,k} + \omega_{i-1,j,k}}{\Delta x_i^2}, \tag{9}$$

where index n related to time level $t_n = n\Delta t$ and (i,j,k) enumerates the grid nodes in x_1, x_2, x_3 directions respectively. The resulting algebraic systems were solved on GPU by the conjugate-gradient method.

In inviscid flow (4), according to the third Helmholtz theorem [21], vorticity lines move as the material fluid particles. Thus the movement of the vortex particles can be described by the infinite set of ordinary differential equations:

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p, t), \qquad \mathbf{x}(0,\boldsymbol{\alpha}) = \boldsymbol{\alpha} \tag{10}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3)$ means the Lagrange coordinates of fluid particles. Solution of Eq. (10) gives the particle-trajectory mapping $\Phi(\cdot,t) : \mathbb{R}^3 \to \mathbb{R}^3; \alpha \to \Phi(\alpha,t) = \mathbf{x} \in \mathbb{R}^3$ that is one-to-one and onto. Incompressibility implies that $det(\nabla_\alpha \boldsymbol{\Phi}(\boldsymbol{\alpha},t)) = 1$

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

$$\mathbf{u}(\mathbf{x},t) = \int \mathbf{K}(\mathbf{x}-\mathbf{x}')\boldsymbol{\omega}(\mathbf{x}',t)d\mathbf{x}' = (\mathbf{K}^*\boldsymbol{\omega})(\mathbf{x},t) \tag{11}$$

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

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

Eq. (11) are the fundamental formulas for direct vorticity methods [14]. 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 $\boldsymbol{\alpha}_i$ is defined by the expression:

$$\alpha_i = \int_{V_p} \omega_i(x_1,x_2,x_3)d\mathbf{x} \approx h^3\omega_i(\mathbf{x}_p), \quad \mathbf{x}_p \in V_p, \quad |V_p| = h^3 \tag{12}$$

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

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{u}(\mathbf{x}_i,t) = \sum_{p=1}^{N_p}\mathbf{K}(\mathbf{x}_i - \mathbf{x}_p)\boldsymbol{\alpha}_p(t) \tag{13}$$

$$\frac{d\boldsymbol{\alpha}_i}{dt} = (\boldsymbol{\alpha}_i(t)\cdot\nabla)\mathbf{u}(\mathbf{x}_i,t) \tag{14}$$

The disadvantage of the direct vortex particle method given by formula (13) is that the method 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 (13) removing the singularity of the kernel $\mathbf{K}$ [17,15,14]. To overcome this difficulty we replaced the Biot–Savart law for velocity calculation with the grid method.

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

$$\mathbf{u} = \nabla \times \mathbf{A} \tag{15}$$

The vector potential $\mathbf{A}$ defines the velocity field with accuracy to some potential field $\nabla\phi$. Adding $\mathbf{A} + \nabla\phi$ we do not change the left side of (15). It is convenient to assume that div$\mathbf{A} = 0$. The vorticity is related to $\mathbf{A}$ as follows

$$\nabla\times\nabla\times\mathbf{A} = \omega = \nabla(\nabla\cdot\mathbf{A}) - \Delta\mathbf{A} \tag{16}$$

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

$$\Delta A_i = -\omega_i, \quad i = 1,2,3 \tag{17}$$

with periodic boundary conditions in each direction.

The Poisson equation can be solved effectively using the numerical grid and finite difference method. Grid solution (17) permits for calculation of the velocity by formula (15).

Subsequently the velocity from the mesh nodes is interpolated onto the particles positions. Such an approach significantly accelerates (~1000 times quicker [6]) the calculations. Algebraic systems obtained from discretisation of the Poisson Eq. (17) were solved by the Multigrid method. The system of Eq. (13) was solved by the Runge–Kutta method of the 4th order. After solution of the Eq. (13) the intensity of the particles was redistributed onto the grid nodes.

We did the redistribution of the intensities of particles onto grid nodes in each time step, before the solution of the Poisson Eq. (17).

It was done using an interpolation:

$$\omega_j = \sum_p \tilde{\alpha}_{p_n}\varphi\left(\frac{\mathbf{x_j} - \tilde{\mathbf{x}}_p}{h}\right)h^{-3} \tag{18}$$

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 [4]

$$\varphi(x) = \begin{cases} (2 - 5x^2 + 3|x|^3)/2 & \text{if } 0 \leqslant |x| \leqslant 1 \\ (2-|x|)^2(1-|x|)/2 & \text{if } 1 \leqslant |x| \leqslant 2 \\ 0 & \text{if } 2 \leqslant |x| \end{cases} \tag{19}$$

For the 3D case, $\varphi = \varphi(x)\varphi(y)\varphi(z)$. The φ satisfies the following moment condition [4]:

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011

$$\sum_j (x - x_j)^k \varphi\left(\frac{x - x_j}{h}\right) = 0 \qquad 1 \leqslant |k| \leqslant m - 1, \qquad m = 3. \tag{20}$$

It says that the kernel (19) is order $m = 3$ [4]. It means that the polynomial up to the order m is exactly represented by the interpolation (18).

3. Remarks on parallel computing

As can be seen from the description of the VIC method, most calculations were related to vortex particles. Calculations of displacement and redistribution for individual particles are independent of one another and can therefore be done in parallel. As there are myriads of vortex particles in the flow (even a few million) a massively parallel environment is desirable. For our calculations we selected Graphics Processing Units (GPUs – also called as a device in [3]). The computer in which device is mounted is called as a host. The GPU have the great computing power-to-cost ratio, but require a quite different approach to programming. Details of implementation of the VIC method on a single GPU can be found in the authors' paper [10].

Implementation of the algorithm for 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 allowed for a numerical mesh not larger than $224 \times 224 \times 224$ nodes. To overcome this limitation we had to write a program capable of using multiple graphics cards. This required distribution and exchange of the data between devices.

Our final implementation of the VIC method was supposed to run on the computer cluster that means a group of computers (called "nodes") used as servers connected to each other through local area network (LAN). Each of the nodes is a stand alone computer and can work on its own. Our target hardware configuration is that each node has more than one GPU (in supercomputing centres one can find nodes with even 8 GPUs).

A stream of instructions along with local storage is called as a process [2]. To modify our program to work on many GPUs we needed to distribute the data among the devices and assure communication between parallel processes.

We decided to use so called hybrid MPI-OpenMP parallel programming (also called multilevel parallel programming) in our code. OpenMP is a standard for parallel programming on systems with shared-memory (in opposite 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. GPUDirect 2.0 technology was used for the communication between GPUs on the same node. It allowed for the transfer of data between two GPUs without copying it to hosts RAM memory. The transfer was done in a non-blocking fashion using CUDA Streams.

Not all of the computational domain has to be transferred between the GPUs. Only a small region (called as a "halo") is needed. To efficiently use streams one has to write a program in such a way that at first calculations are carried out on "halo" region. Then one stream transfers data to the host memory and the second performs calculations for the remaining part of the computational domain. In this way the transfer of the data and computations can be done concurrently.

The halo region may be scattered across devices memory. If so one have to manually select pieces of the data, copy them into a continuous memory space and after copying between processes put it in the right place on a new device. In the presented work we decomposed the computational domain only in one direction (z-axis). In this way the halo region was a continuous memory area. In our computation the data had to be sent in three different cases. First was the solution of the algebraic system of equations by iterative methods (multigrid and conjugate gradient). In this case the halo region size was dependent on the numerical stencil used for the decomposition of the solved equations. In this paper we used the second order central difference scheme in space. This means that to compute the value in the next iteration in the given point we needed information only about the neighbouring nodes and in this case the halo region was of size 1. The second case in which the halo region had to be transferred was the displacement of the particles. It was done with the Runge–Kutta 4th order method. During the displacement of the particles the velocity and its derivatives had to be interpolated onto the particles instantaneous positions. As the 3rd order interpolation (see (19)) and central difference schemes was used the halo region was of size 3. This transfer has to be done once in each time step. The third case was connected to the redistribution of particle intensities to the grid nodes. The data related to positions and intensities of particles had to be sent. The data about all of the particles that might have had an influence on the nodes in different domains was transferred. As we used interpolation kernel (19) and assured that the particle did not travel more then one grid step size per time step it was enough to send particles that started their movement at the domain boundary and two of its neighbours (halo region was of size 3).

When one is using many devices in computation the data has to be transferred between them. This takes time and slows down computation. The limiting factor in our test on a single GPU was the amount of the RAM memory on a single device. Our main interest was to enlarge the number of the grid nodes to rise the precision of the calculation and to widen computational domain. To characterize the loss of the computational speed after introducing additional GPU units we used coefficient defined as

$$E_w(p, n) = \frac{T(1, n)}{T(p, n)} \tag{21}$$

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

This coefficient is called as a weak scaling and is used when there is a constant amount of data per one graphics card and number of graphics cards is growing. This simulates a situation in which the test case will be too large to fit on one GPU. Each graphics card 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 many GPUs case would be the same as for a single card. Unfortunately, this is not always possible.

Test were done on a computer with four NVIDIA GTX570 graphics cards (3 GB each). For the $224 \times 224 \times 224$ case (the biggest case to fit on a single GPU with 3 GB of memory) the efficiencies for the weak scaling can be seen in Table 1 respectively.

As one can see going from 1 to 2 GPU there is a weak scaling efficiency of 0.91. This quite a good result. Worse result was obtained using 4 GPUs. This may be caused by the fact that the

Table 1
Weak scaling. Duration of 5 time steps of the VIC method.

No. of GPUs	Time (ms)	Weak scaling efficiency
1	92,090	–
2	101,648	0.91
4	159,805	0.58

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011

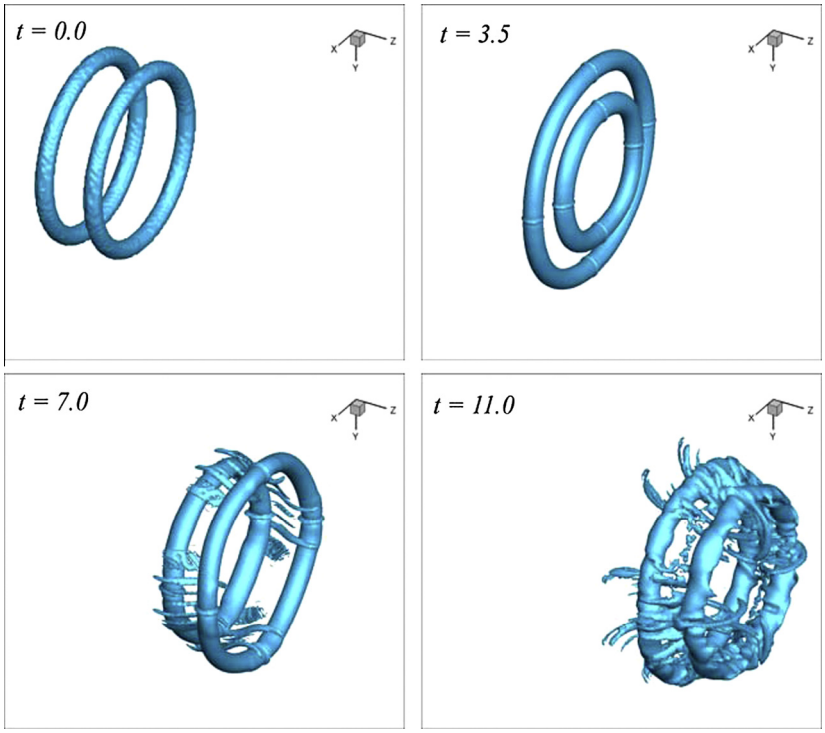

Fig. 1. Isosurface plot $|\omega| = 0.2\omega_0$ of leapfrogging in the inviscid fluid.

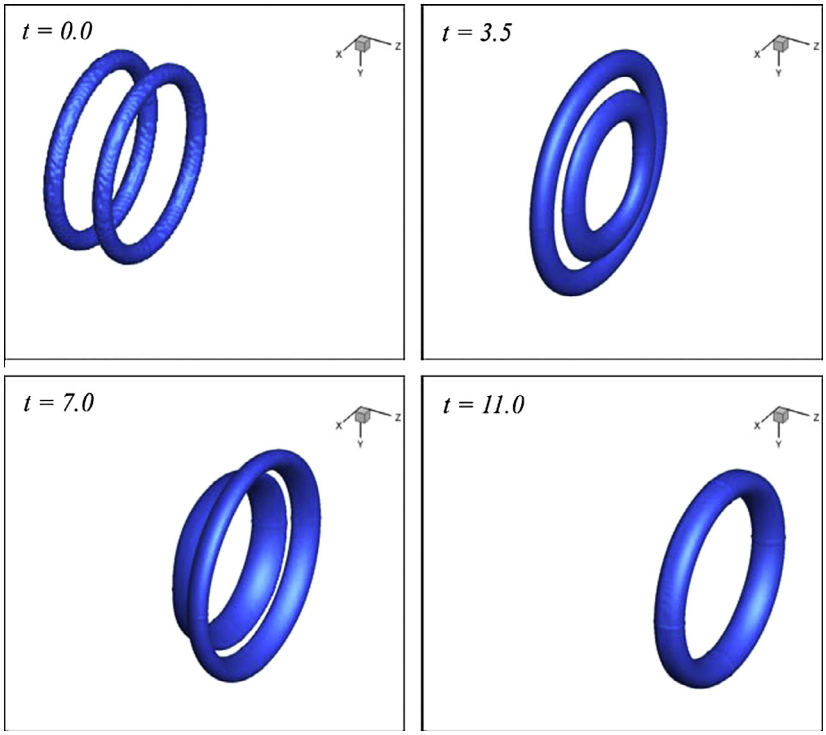

Fig. 2. Isosurface plot $|\omega| = 0.15\omega_0$ of leapfrogging in the viscous fluid.

motherboard used in our computer is capable of transferring data to at most 2 GPUs with the maximum rate given by the PCI Express $\times 16$ slot (theoretical peak of 8 GB/s and about 6 GB/s in tests). When 4 GPUs are transferring data altogether the transfer is only with half of its maximum rate to each GPU (theoretical peak of 4 GB/s and about 3 GB/s in tests). This shows that different hardware could improve results.

The total amount of computational time for the test case of the head-on collision on a computational grid of $256 \times 256 \times 256$ nodes and 2000 time steps on our cluster using 2 GPUs is about 9 h.

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011

The performance results shown above are not perfect. In our opinion one of the limiting factors was the used hardware which introduced long data transfer times for the configurations with 4 GPUs. Our implementation of the Vortex-in-Cell method allows for the computations on any number of computational nodes having GPUs. Thanks to this we can conduct computations even in supercomputing centres where current problems with hardware will not be present.

4. Formulations of the numerical test problems and numerical results

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 [9]. 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 [22]. The breaking and rejoining of vortex lines may be a fundamental process in the evolution of three-dimensional vortices and the mechanics of turbulence [16].

We intend to show that vortex particle method is capable of reproducing results from the real life experiments. In the previous article [11] the authors showed good agreement with numerical studies conducted by other numerical methods.

As the test cases for computation on many graphics cards, we have chosen to investigate the dynamics of the motion of two vortex rings undergoing so called leapfrogging or a head-on collision.

Distribution of vorticity in the cross section was assumed as:

$$\omega(r) = \omega_0 e^{-\frac{r^2}{a^2}} \tag{22}$$

where ω_0 and a are given constants. Based on this the total circulation Γ and the Reynolds number were calculated

$$\Gamma = \int_0^a \omega(r) 2\pi r \, dr, \qquad Re_\Gamma = \frac{\Gamma}{\nu} \tag{23}$$

where ν is kinematic viscosity coefficient.

In every test case presented the numerical mesh size was 256^3 and the domain of computation was a box $[20 \times 20 \times 20]$ with periodic boundary conditions in all directions.

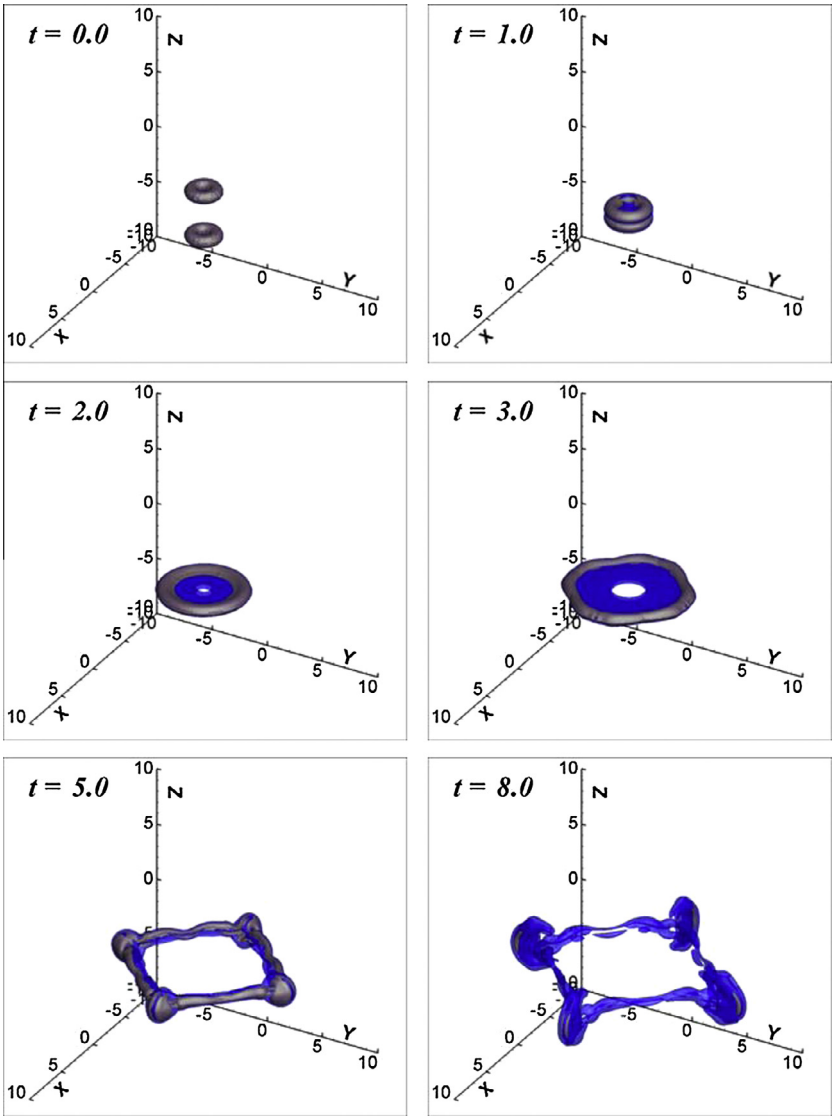

Fig. 3. Isosurface plot $|\omega| = 0.05\omega_0$ (blue) and $|\omega| = 0.2\omega_0$ (yellow – look like grey when covered with blue) of collision of two vortex rings. $Re_\Gamma = 1000$.

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011

4.1. Leapfrogging

In this section the numerical simulation results for leapfrogging of two vortex rings are presented. Leapfrogging strongly depends on initial position of the vortex rings and sometimes may not take place [12].

During this 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.

4.1.1. Inviscid fluid

First test was the leapfrogging in the inviscid fluid. The parameters in the Eqs. (22) and (23) were $\omega_0 = 22.38$, $a = 0.15$, $R = 1.5$, $\Gamma = 1.0$.

As one can see in Fig. 1 the rings undergo two consecutive slip-throughs. Very similar result, for a real life experiment, can be found in [12]. Although the rings become fuzzy they do not merge into one ring as this is impossible in the inviscid case.

4.1.2. Viscous fluid

Second test was conducted for the viscous fluid. The parameters in the Eqs. (22) and (23) were the same as for the inviscid case and $Re_\Gamma = 1000$.

In this case vortex rings were able to undergo one slip-through and then merged to form a single ring. We think that this is due to the fact that vortex ring cores were diffused during the movement and the interaction between them was too weak to move them apart (see Fig. 2).

4.2. Collision of two vortex rings $Re_\Gamma = 1000$

This test shows the head–on collision of the two vortex rings. The experimental results were published by Lim and Nickels in [13] and also very clearly on the web side of Lim [1] http://serve.me.nus.edu.sg/limtt/.

In this case the parameters were as follows $\omega_0 = 20$, $a = 0.5$, $R = 1.0$, $v = 0.01$, $Re_\Gamma = 1000$. The results can be seen in Fig. 3.

When two rings approach each other the velocities induced by the them cause both rings to grow in diameter and become flat. At first one can notice that the vorticity distributions around the axis of symmetry create the thin sheet that was called by Lim as a membrane (frame $t = 2$ in Fig. 3). As was reported in [13] when the size of the rings are approximately four times their initial diameter, a symmetrical instability in the form of azimuthal waviness begin to developed (frame $t = 3$ in Fig. 3). That instability lead to the creation on the circumference, four bulges of the vorticity. Our results are related to the pictures presented on the web side of Lim for Reynolds number smaller then 1000. In Lim's experimental results the number of the bulges is five times higher than in our computations. It is hard to compare Lim's results to our computations exactly. We conducted computations on a finite size cube $20 \times 20 \times 20$ with the periodic boundary conditions. Lim related the Reynolds number to the diameter of the hole by which the rings were produced and not to the circulation of the rings as

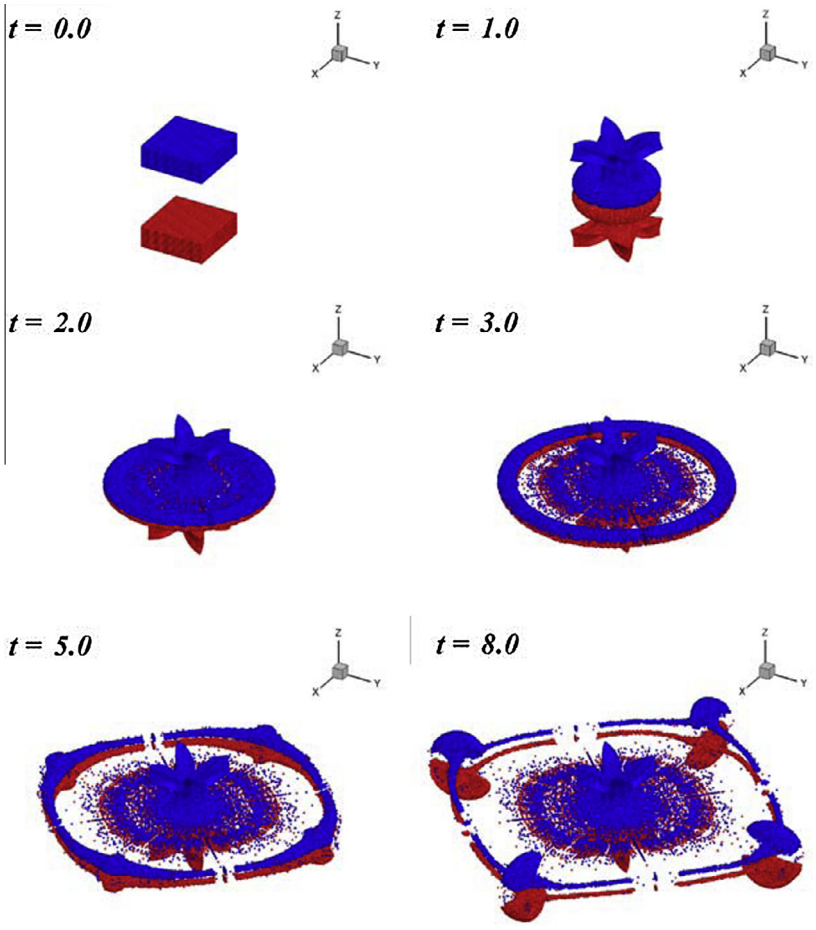

Fig. 4. Collision of two vortex rings visualized by the passive marker. On the frame $t = 0$ the vortex rings are merged inside of the cuboids.

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011

we did. Smaller number of bulges on circumference of vortex structure and rectangular shape of the structure in the end of our computation may results from that. Also the size of our computational domain and periodic boundary conditions impact on our results.

It is known that vorticity is not carried by the fluid particles in the viscous flow and the vorticity is not prefect means for visualization of the flow. Due to this, in order to better compare our results to the real flow visualization we used the passive markers carried by the fluid. The rings we merged in boxes of passive markers (coloured with the initial position: red – lower ring, blue – upper ring). The size of the cuboids was equal to the diameter of the vortex rings and their thickness. Inside of the each box we put a grid of $100 \times 100 \times 50$ nodes and at each of the node a passive marker was put. It is clear that some passive markers laid outside of the rings. Our numerical results were presents in Fig. 4. One can see that in frame $t = 1$ a tail created by the particle initially placed outside of the rings is formed. In the frame $t = 3$ one can see formation of the membrane. The structure that can be seen in the frame $t = 5$ very much resembles the experimental pictures of Lim [1]. From the frame $t = 8$ it is clear that reconnection did not take place. The particles did not mixed. The reconnection phenomena that would appear in the vorticity bulges could be responsible for the creation of the small rings on circumference as was reported in [13].

5. Closing remarks

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 rings. In spite of the simple geometry the vortex rings evolution provide various fascinating phenomena like leapfrogging, head-on collision of the rings as was reported by [13]. We showed that our implementation of vortex particle method is capable of reproducing result obtained in experiments.

Nowadays it is not difficult to notice that the computational power of a single processor stopped rising. 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 get the results faster. The GPUs are also quite cheap and easily accessible.

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. Properly used GPUs (memory management, parallel algorithms, etc.) allows programs to be executed much faster at relatively low cost. Future work is to run our program on a supercomputer.

References

[1] Lim TT. Webpage 2011. <serve.me.nus.edu.sg/limtt/>.
[2] Cyberinfrastructure tutor (ci-tutor); 2012. <www.citutor.org>.
[3] Nvidia cuda c programming guide; 2012. <www.nvidia.com/cuda>.
[4] Cottet GH, Koumoutsakos PD. Vortex methods: theory and practice. Cambridge University Press; 2000.
[5] Cottet GH, Michaux B, Ossia S, VanderLinden G. J Comput Phys 2002;175:1–11.
[6] Cottet GH, Poncet P. J Comput Phys 2003.
[7] Etancelin JM, Cottet GH, Picard C, Perignon F. Particle method on gpu, 2013. CANUM 2012. In: ESAIM Proceedings [in press]. <http://hgpu.org/?p=8860>.
[8] Holden H, Karlsen KH, Lie KA, Risebro WH. Splitting methods for partial differential equations with rough solutions. Eur Math Soc 2010.
[9] Kida S, Takaoka M. Annu Rev Fluid Mech 1994;26:169–89.
[10] Kosior A, Kudela H. Comput Fluids 2012.
[11] Kudela H, Kosior A. Procedia IUTAM 2013;7:59–66.
[12] Lim TT. Phys Fluids 1997;9:239–41.
[13] Lim TT, Nickels TB. Nature 1992;357:225–7.
[14] Majda AJ, Bertozzi AL. Vorticity and incompressible flow. Cambridge University Press; 2002.
[15] Meiburg E. Fluid vortices, fluid mechanics and its applications. Kluwer Academic Publishers; 1995. p. 651–85.
[16] Melander VM, Hussain F. Phys Fluids A 1989;1:633–6.
[17] Puckett EG. Incompressible computational fluid dynamics trends and advances. Cambridge University Press; 1993. p. 335–407.
[18] van Rees WM, Leonard A, Pullin D, Koumoutsakos P. J Comput Phys 2011;230:2794–805.
[19] Rossinelli D, Bergdorf M, Cottet GH, Koumoutsakos P. J Comput Phys 2010;229:3316–33.
[20] Winckelmans G. Vortex methods. In: Stein E, De Borst R, Hughes TJ, editors. Encyclopedia of computational mechanics, vol. 3. John Wiley and Sons; 2004.
[21] Wu JZ, Ma HY, Zhou MD. Vorticity and vortex dynamics. Springer; 2006.
[22] Zabusky NJ, Melander MV. Physica D 1989;37:555–62.

Please cite this article in press as: Kosior A, Kudela H. The 3D vortex particle method in parallel computations on many GPUs. Comput Fluids (2013), http://dx.doi.org/10.1016/j.compfluid.2013.10.011