TASK QUARTERLY vol. 18, No 1, 2014, pp. 67–80

SMOOTHED PARTICLE HYDRODYNAMICS SIMULATIONS USING GRAPHICS PROCESSING UNITS

KAMIL SZEWC

Institute of Fluid-Flow Machinery, Polish Academy of Sciences
Fiszera 14, 80-231 Gdansk

(Received 10 December 2013; revised manuscript received 20 January 2014)

Abstract: Smoothed Particle Hydrodynamics (SPH) is a fully Lagrangian, particle-based technique for fluid-flow modeling. As a gridless method, it appears to be a natural approach to simulate multi-phase flow with complex geometries. Since SPH involves a large set of short-range particle-particle interactions, numerical implementations present a high degree of spatial data locality and a significant number of independent computations. Therefore, the numerical code can be easily written in a massively parallel manner. The main purpose of this study is to discuss the issues related to the implementation of the SPH method for computation using Graphics Processing Units (GPU). The study is supported by two-dimensional validation cases: the lid-driven cavity and oscillation of a droplet. The obtained results show a good accuracy of the method, as well as, high numerical efficiency of its GPU implementation.

Keywords: fluid dynamics, CFD, particle methods, SPH, GPU calculations

1. Introduction

The Smoothed Particle Hydrodynamics method (SPH) is a fully Lagrangian, particle based approach for fluid-flow simulations. One of its main advantages over the Eulerian techniques is no need of a numerical grid. Therefore, there is no necessity to handle the interface shape as in grid-based methods. Thus, the SPH method is increasingly used for hydro-engineering and geophysical applications involving free-surface and multi-phase flows. One disadvantage of the SPH approach over the Eulerian methods is the numerical efficiency. However, in many cases involving complex geometry, the "human" time needed to create a computational grid can be so long, that it can be more time- and cost-efficient to perform calculations using mesh-free approaches, such as SPH. In addition, in recent years new techniques allowing numerical simulations to be performed using Graphics Processing Units (GPU) have been developed. The massive parallel

computation capability of modern graphics cards allows simulations of large systems to be performed using cheap desktop computers. Particle methods are easy to write in a parallel manner, hence they are suitable for calculations on GPUs.

The main purpose of this work is to discuss issues related to the acceleration of the SPH calculations using GPU devices. The paper is divided into seven sections. The first section is this introduction. In Sections 2 and 3 the fundamental concepts of the SPH methods are briefly recalled and the SPH form of governing equations is introduced. Section 4 contains information about the GPU implementation issues and used algorithms. In Section 5 the SPH results of the lid-driven cavity, as well as, droplet oscillation tests are presented and compared with the reference data. Section 6 is devoted to discussing the efficiency of the GPU implementation. The purpose of the last section is to summarize the obtained results.

2. Smoothed Particle Hydrodynamics

The main idea behind the SPH approach is to introduce kernel interpolants for flow quantities so that fluid dynamics is represented by a set of particle evolution equations, cf. [1] for a review. Then, three approximations are made to devise practical formulations.

The first formulation is an interpolation of field quantities at a point. To construct it, we utilize an integral interpolant $\widehat{A}(\mathbf{r})$ of any field $A(\mathbf{r})$

$$\widehat{A}(\mathbf{r}) = \int_{\Omega} A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}', \tag{1}$$

where the integration is over all the domain Ω. $W(\mathbf{r},h)$ is a weighting function (kernel) with the parameter h (smoothing length) describing the range of the kernel. Generally, the kernel should have a symmetrical form

$$W(\mathbf{r},h) = W(-\mathbf{r},h), \tag{2}$$

satisfy the limit condition

$$\lim_{h\to 0} W(\mathbf{r},h) = \delta(\mathbf{r}), \tag{3}$$

where $\delta(\mathbf{r})$ is the Dirac delta distribution, it should be sufficiently smooth and normalized, so that

$$\int_{\Omega} W(\mathbf{r},h)d\mathbf{r} = 1. \tag{4}$$

Taking into consideration the computational effort and proper implementation of the boundary conditions, it is worth using kernels having a compact support. Since there are many possibilities for the choice of $W(\mathbf{r},h)$ in order to avoid kernel artefacts such as particle clustering [2], after [3], we use the Wendland kernel [4] in the form (in 2-D)

$$W(\mathbf{r},h) = \frac{7}{4\pi h^2}\begin{cases} \left(1-\frac{q}{2}\right)^4(2q+1) & \text{for } |\mathbf{r}| \le 2h, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where $q = |\mathbf{r}|/h$. In 3D, the only difference is the normalization constant equal to $21/16\pi h^3$. An optimal choice of the kernel and the simulation parameters has been discussed in [3].

The second approximation of the SPH technique is the discretization of media. It is achieved by a fine-grained representation (particles) of fluid in the flow domain, where each particle carries the properties of the field. Then, the integral interpolant $\widehat{(\cdot)}$, Equation (1), becomes a summation interpolant $\langle\cdot\rangle$

$$\langle A\rangle(\mathbf{r}) = \sum_b A(\mathbf{r}_b) W(\mathbf{r}-\mathbf{r}_b, h)\Omega_b, \tag{6}$$

where $\mathbf{r}_b$ and Ω_b denote the position and volume of the particle b. The SPH task involves the computation of the interpolant at each particle, cf. Figure 1, so that Equation (6) may be rewritten into the form

$$\langle A\rangle_a = \sum_b A_b W_{ab}(h)\Omega_b, \tag{7}$$

where $\langle A\rangle_a = \langle A\rangle(\mathbf{r}_a)$, $A_a = A(\mathbf{r}_a)$ and $W_{ab}(h) = W(\mathbf{r}_b - \mathbf{r}_a, h)$.

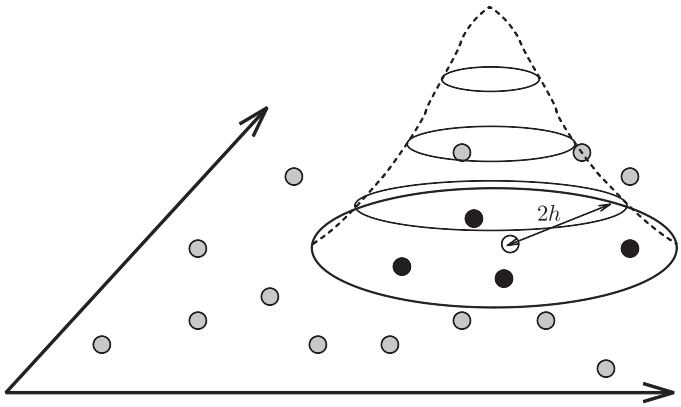

Figure 1. A set of neighboring particles in SPH; due to the finite support of the (compact) kernel, it is only black particles that interact with the white one

An additional advantage of SPH is revealed when the differentiation of fields is considered. In accordance with (1), the gradient of $A(\mathbf{r})$ has the form

$$\widehat{\nabla A}(\mathbf{r}) = \int_\Omega \nabla A(\mathbf{r}') W(\mathbf{r}-\mathbf{r}', h) d\mathbf{r}'. \tag{8}$$

Taking advantage of the integration by parts and utilizing the kernel symmetry, after discretization, the above expression can be further transformed to

$$\langle \nabla A\rangle_a = \sum_b A_b \nabla_a W_{ab}(h)\Omega_b. \tag{9}$$

Since the differentiation operator acts only on the kernel, the gradient of the field is dependent only on the values of the field at particles. Higher derivatives can be obtained in a straightforward manner, nonetheless, due to the accuracy and

efficiency requirements, the commonly used form is built as a combination of the finite difference approach and the SPH approximation [5].

The third SPH approximation consists in assuming that the field value A_a at a point and its SPH approximation $\langle A\rangle_a$ are equal:

$$\langle A\rangle_a \approx A_a, \tag{10}$$

which amounts to neglecting the effects of scales which are typically smaller than h and which are smoothed out.

3. Governing equations

A full set of governing equations for an incompressible viscous flow is composed of the Navier-Stokes (N-S) equation

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{\varrho}\nabla p + \frac{1}{\varrho}(\nabla\cdot\mu\nabla)\mathbf{u} + \mathbf{f}, \tag{11}$$

where ϱ is the density, $\mathbf{u}$ the velocity, t the time, p the pressure, μ the dynamic viscosity and $\mathbf{f}$ external acceleration, and the continuity equation

$$\frac{d\varrho}{dt} = -\varrho\nabla\cdot\mathbf{u}, \tag{12}$$

that for $\varrho = const$ takes the form

$$\nabla\cdot\mathbf{u} = 0. \tag{13}$$

The governing equations can be expressed in the SPH formalism using Equations (6) and (9). However, it is important to note that it can be done in many different ways, cf. [6] and [7]. In this paper, we consider the formulation proposed by Hu and Adams [8]. To avoid any inconsistency of the density and velocity fields caused by the use of the continuity equation, for details cf. [7], Hu and Adams decided to take advantage of the density definition in the form

$$\varrho_a = m_a\sum_b W_{ab}(h) = \frac{m_a}{\Theta_a}. \tag{14}$$

The N-S pressure term which is variationally consistent with Equation (14) has the form (cf. [9] for details)

$$\left\langle\frac{\nabla p}{\varrho}\right\rangle_a = \frac{1}{m_a}\sum_b\left(\frac{p_a}{\Theta_a^2}+\frac{p_b}{\Theta_b^2}\right)\nabla_a W_{ab}(h), \tag{15}$$

while the N-S viscous term can be written as (cf. [8] and [10])

$$\left\langle\frac{1}{\varrho}(\nabla\cdot\mu\nabla)\,\mathbf{u}\right\rangle_a = \frac{1}{m_a}\sum_b\frac{2\mu_a\mu_b}{\mu_a+\mu_b}\left(\frac{1}{\Theta_a^2}+\frac{1}{\Theta_b^2}\right)\frac{\mathbf{r}_{ab}\cdot\nabla_a W_{ab}(h)}{r_{ab}^2+\eta^2}\mathbf{u}_{ab}, \tag{16}$$

where $\eta = 0.01h$ is a small regularizing parameter added to avoid potential singularities.

In the SPH method there are many ways to ensure the incompressibility condition. In the present work, due to the easiness of implementation onto GPU

units, it was decided to use the Weakly Compressible SPH (WCSPH) method. The WCSPH method is implemented by adding the artificial equation of state to the governing equations

$$p = B\left[\left(\frac{\varrho}{\varrho_0}\right)^{\gamma} - 1\right], \tag{17}$$

where $B = c^2\varrho_0/\gamma$, the reference density ϱ_0, the numerical sound speed c and γ are suitably chosen to reduce the density fluctuations down to the demanded level.

The boundary condition is introduced by the so-called Ghost-Particle method, which is similar to the well-known Classic Image Problem in the electrostatics. The method consists in creation of modified images of particles located near the boundary to result in proper physical fields near the wall. For details, cf. [3] and [11].

4. GPU implementation issues

4.1. Multi-threading

A typical CPU supports one or two threads per core (2–16 cores in modern devices). In contrast, a typical GPU has from 8 up to 192 streaming processors (cores) allowing 8 to 192 of threads to run concurrently. For a detailed description of the NVIDIA CUDA architecture see [12] and [13]. Streaming processors are organized in streaming multiprocessors. Modern GPU cards, like NVIDIA Titan, have even 15 streaming multiprocessors (2880 cores). A multi-threaded program is partitioned into blocks of threads which run independently from each other, so that a GPU can spread blocks between multiprocessors resulting in faster execution. Blocks of threads are organized into a one-, two- or three-dimensional grid, cf. Figure 2.

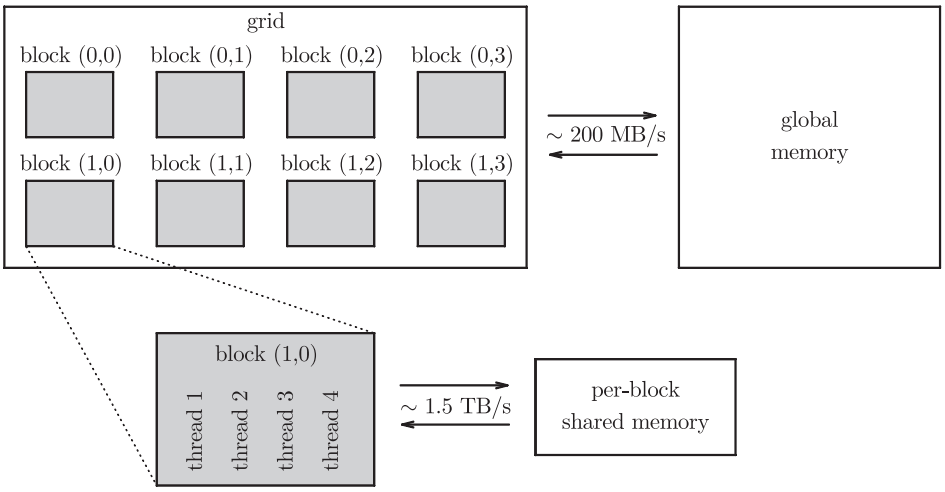

Figure 2. Grid, block and thread organization in the NVIDIA CUDA programming model; memory-hierarchy allows threads within a single block to use the fast shared memory; all threads share access to the global memory

Since all threads in a block have to reside in the same streaming multiprocessor and must share limited memory resources, there is a hardware limit of the number of threads per block (currently 768–2048). It is only threads within a single block that can cooperate by synchronizing the execution and sharing data using shared memory (cf. Section 4.4). The proper choice of the number of threads per block depends on both the GPU device and the running algorithm. The influence of this parameter on the performance of our SPH implementation on the GPUs available to us is described in Section 6. Summarizing, the complex architecture of modern GPU devices makes GPU programming much harder than programming for classical CPUs. Well-written GPU algorithms should be well-scalable with respect to the number of threads, even at the expense of lower performance of a single thread. Since the SPH method involves a large set of independent particle-particle interactions its numerical code can be easily designed to efficiently run on GPU by performing interactions of each particle within an independent thread (the number of threads is equal to the number of particles). It is important to note, that no thread synchronization is necessary for the SPH interaction procedure, therefore, the efficiency can be well-scalable with respect to the number of particles.

4.2. Searching for neighboring particles

A primary way to increase the efficiency of the SPH method is to use the kernels of the compact support form, and then, to search for and interact only with the neighboring particles which are close enough to have non-zero impact on the result. For this purpose, the computational domain is subdivided to form an auxiliary grid. In our code, the uniform grid, which is the simplest possible spatial subdivision, has been implemented. For simplicity, we use a grid where the cell size is the same as the range of the kernel ($2h$ for the Wendland kernel). This means that each particle can interact only with a limited number of particles located in the neighboring 9 cells in 2D (27 in 3D), cf. Figure 3.

The particles-grid relationships are generated at each time step. it is possible to perform incremental updates to the particle-grid structure, but, due to the insignificant effect of a generation from the scratch procedure on the computational time, we decided to implement the simplest scheme. In the classical CPU algorithms, the particles-grid relationships are held in the form of a linked-list data structure. Each node is composed of data and a reference to the next node in the sequence. The linked-lists allows for efficient insertion ($O(1)$ if the last element is known) of nodes from any position in a sequence. For the SPH method, the principal benefit of using linked-lists is quick access to information about the location of particles in cells.

4.3. Single precision calculations

Since GPU cards were designed to accelerate the creation of images in a frame buffer to output them onto a display, the double precision for such calculations was not needed. Therefore, most of the desktop GPU cards were designed

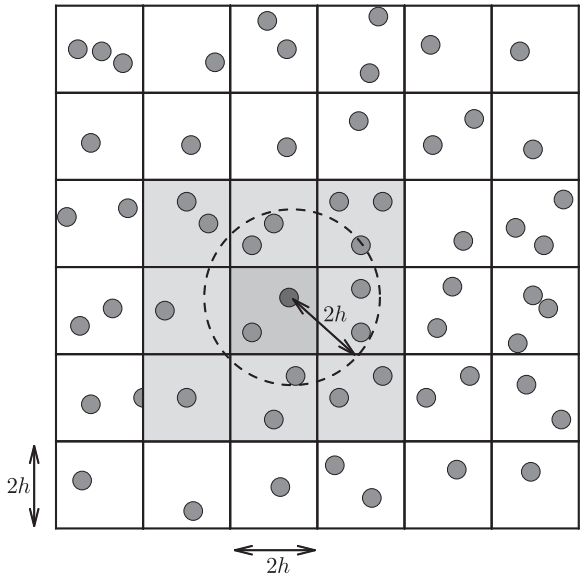

Figure 3. Particle interaction scheme in two-dimensional SPH; the highlighted particle may interact only with particles located in the neighboring (colored) cells

to perform fast calculations with single precision only. There is a possibility to perform double precision calculations, however, typically it is 16 times slower. There are some GPU devices supporting fast double precision, e.g. NVIDIA Tesla series. Although they are expensive, double precision are still much slower than single precision calculations. However, it is important to note that lesser accuracy related to this issue is not a problem for the SPH method. The numerical errors related to the kernel shape, random distribution of particle positions and finally approximations are much higher than the truncation error. In practice there are no important differences between single and double precision SPH calculations.

4.4. GPU memory

One of the important advantages of GPU over CPU computation is its potential memory bandwidth and latency. There are five different storage types on a GPU: registers ($\sim 8\,\mathrm{TB/s}$), shared memory ($\sim 1.5\,\mathrm{TB/s}$), texture memory ($\sim 200\,\mathrm{MB/s}$), constant memory ($\sim 200\,\mathrm{MB/s}$) and global memory ($\sim 200\,\mathrm{MB/s}$). Well written GPU software should use registers to the largest possible extent. However, the problem is that the number of registers is highly limited, hence, it is necessary to use another type of memory, unfortunately with much more inefficient access, to implement interaction (exchange of data) between threads. In such a case, the best choice is to use shared memory. The problem here is that shared memory has small capacity and, additionally, each streaming multiprocessor has its own shared memory, thus, threads which run on streaming processors belonging to different streaming multiprocessors cannot share data using this kind of memory, cf. Figure 2. From the point of view of the NVIDIA

CUDA language, shared memory is restricted to threads belonging to one block of threads. The only types of memory shared by all threads are texture, constant and global memory, however, their bandwidth and latency are much smaller. Thus, the best solution is to accordingly organize the exchange between different types of memory to use registers and shared memory as commonly as it is possible. Hopefully, in the SPH method, the artificial mesh used for searching of neighboring particles is a natural constraint allowing a good organization of memory on a GPU. Interaction between particles occurs only for those particles which are located in the neighboring cells. Therefore, ordering memory to run interaction of particles within one streaming multiprocessor using shared memory can have a significant impact on performance. It is important to note that different GPU devices have a different size of the shared memory, as well as, a different number of streaming processors and multiprocessors. It results in the necessity to adjust the ordering procedure to the given GPU device. In our code we only load particle-grid hash data (cf. Section 4.2) into the shared memory to look at the hash values of the neighboring particle without loading two hash values per thread. Another problem is a small amount of the GPU's global memory comparing to the memory available in a CPU. High-end desktop GPUs have 4–6 GB memory. An expensive Tesla K40 designed especially for scientific computations has only 12 GB memory. Such amount of memory may be not enough for complex 3D fluid-flow problems involving hundreds of millions of particles. Therefore, for such systems, there is a necessity to use multi-GPU solutions; however, the communication between GPUs can highly decrease the overall performance.

5. Numerical results

5.1. Lid-driven cavity

A two-dimensional lid-driven cavity is a common test for numerical algorithms for viscous flows with the presence of no-slip boundaries. It involves a fluid of density ϱ inside a square $(L \times L)$ box where the upper boundary moves horizontally with the constant velocity $\mathbf{u}_w$. Although the geometry is very simple, there is no analytical solution. For validation purposes, it was decided to compute the lid-driven cavity flow at $Re = |\mathbf{u}_w|L/\nu = 100$,1000 and 10000. All variables are suitably non-dimensionalized with L, $|\mathbf{u}_w|$ and ϱ. The flow results are compared with the numerical calculation performed on a fine-grid using the Eulerian solver by Ghia *et al.* [14]. We decided to use $N = 4096$ particles in the domain for $N = 100$, $N = 65536$ for $\mathrm{Re} = 1000$ and $N = 262144$ for $N = 10000$, while $h/\Delta r = 2$, where Δr is a measure of the inter-particle distance (at the initial state). Figure 4 presents the steady-state velocity fields obtained via projection of particle velocities into a regular (50×50) grid.

The vortex moves toward the centre of the domain with an increase in the Reynolds number. The steady-state velocity profiles are presented in Figures 5. All the three considered cases are in good accordance with the reference data. The numerical performance of GPU calculations is discussed in Section 6.

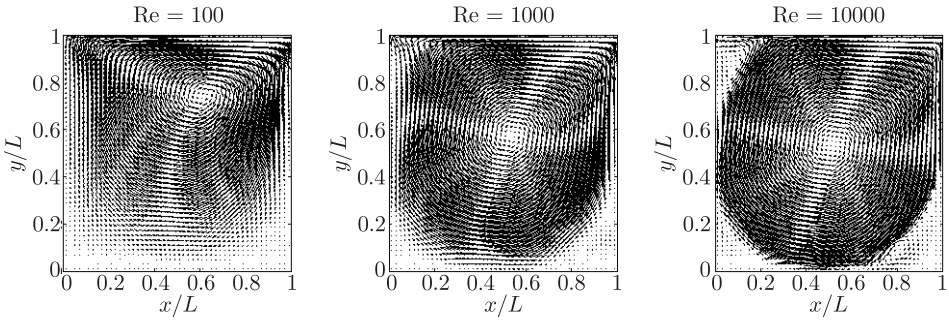

Figure 4. Lid-driven cavity steady-state velocity at $Re = 100$, 1000 and 10000; velocity fields obtained via projection of particle velocities into a regular 50×50 grid

5.2. Oscillating droplet

One of the most common two-dimensional dynamic multiphase test cases is a circular droplet oscillating under the action of the surface tension force. Here, we decided to simulate oscillation of a droplet corresponding to an air-liquid system: the density and dynamic viscosity ratio between the droplet and the surrounding gas were $\varrho_D/\varrho_G = 1000$ and $\mu_D/\mu_G = 100$, respectively. The simulations were performed in a square box of size L. The droplet of radius $R = 0.1875L$ was placed at the centre of the domain. The initial perturbation was given by the divergence-free velocity field in the form

$$\begin{aligned} u_x &= u_0 \frac{x}{r_0}\left(1 - \frac{y^2}{r_0 r}\right)\exp\left(-\frac{r}{r_0}\right), \\ u_y &= u_0 \frac{y}{r_0}\left(1 - \frac{x^2}{r_0 r}\right)\exp\left(-\frac{r}{r_0}\right), \end{aligned} \tag{18}$$

where $u_0 = 4\sqrt{R^3 \varrho_D/\sigma}$, $r_0 = 0.25R$, while r is the distance to the droplet center.

Lamb [15] showed that the frequency of oscillations of a two-dimensional droplet is given by

$$\omega_n = \sqrt{\frac{(n^3 - n)\sigma}{\varrho_D R^3}}, \tag{19}$$

where n is the mode of oscillation. For $n = 2$, corresponding to our case, the theoretical oscillation period is

$$T = 2\pi\sqrt{\frac{R^3 \varrho_D}{6\sigma}}. \tag{20}$$

Simulations were performed for $N = 120 \times 120$ particles, initially homogeneously distributed in the domain, $h/\Delta r = 2$ and the Wendland kernel (5). We decided to use the Continuum Surface Force technique as a surface-tension model, cf. [16] for details. Surface tension can be described by the Weber number $\mathrm{We} = \varrho_D R u^2/\sigma$ and the capillary number $\mathrm{Ca} = \mu_D u/\sigma$. Here, we chose $\mathrm{Ca}^2/\mathrm{We} = \mu_D^2/R\sigma\varrho_D = 0.055$. The capillary number, based on u_0, was 0.065. The evolution of particle distributions is presented in Figure 6. The results are suitably

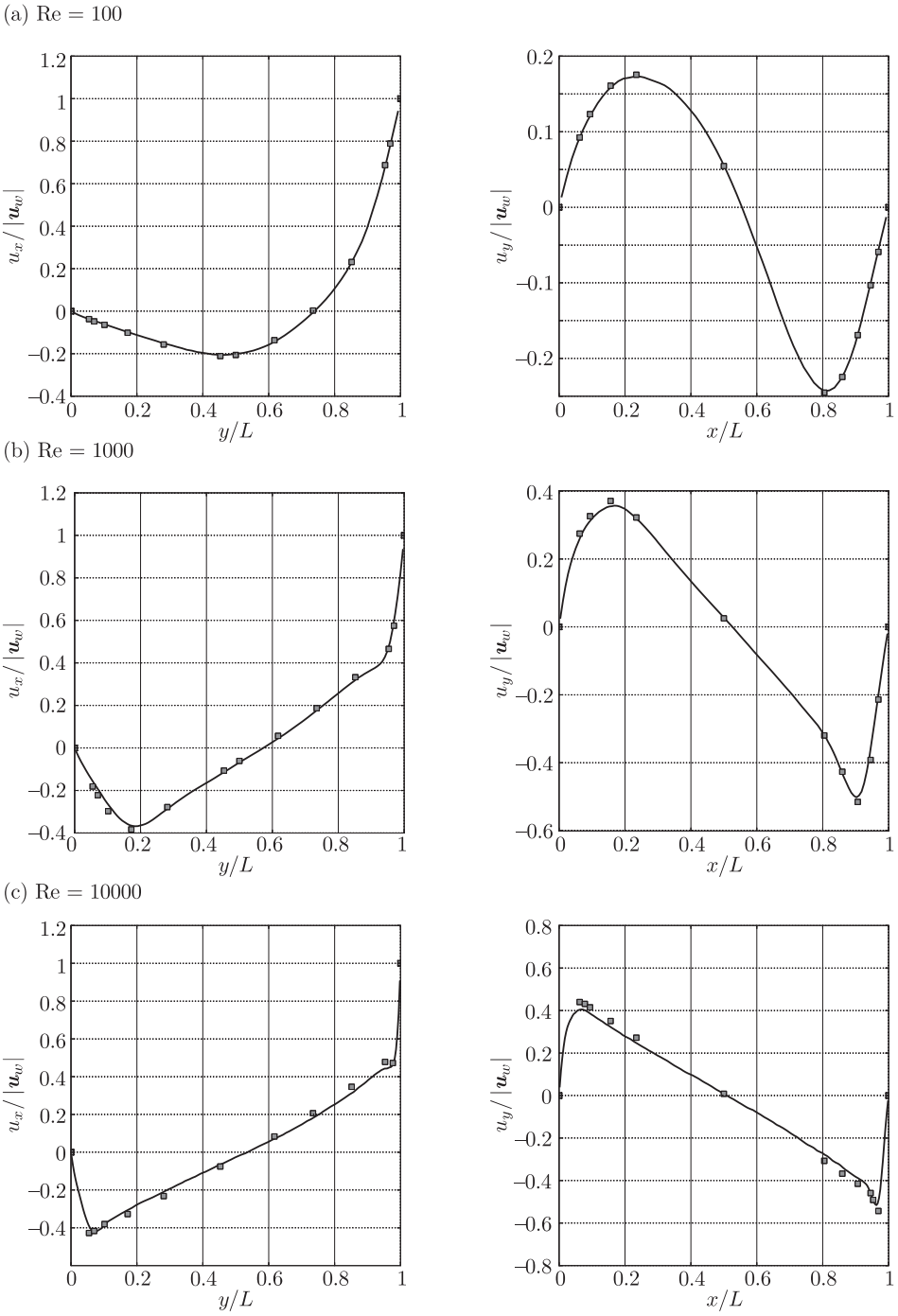

Figure 5. Lid-driven cavity velocity profiles at $Re = 100$, 1000 and 10000: (left) u-velocity along vertical line and (right) v-velocity along horizontal line, both through the geometric centre of cavity; the reference results (gray squares) from [14]

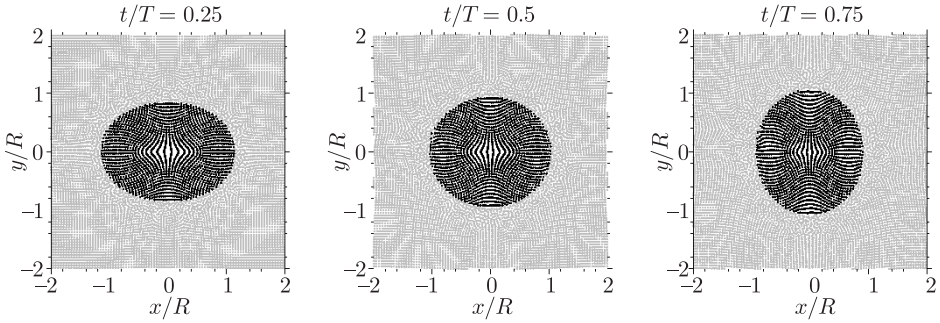

Figure 6. Evolution of particle positions in droplet oscillation test ($N = 120 \times 120$)

normalized by the droplet radius R in space and the theoretical value of period T in time.

To validate the correctness of the SPH GPU implementation, we compared the oscillation period calculated using the SPH method with the theoretical result (20). Figure 7 shows the droplet oscillation period as a function of the surface tension coefficient. The obtained results show good agreement with the reference data.

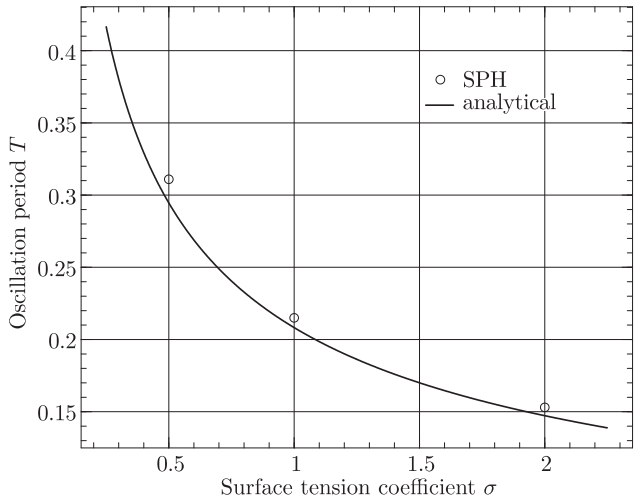

Figure 7. Droplet oscillation period obtained from SPH simulation as a function of the surface tension coefficient; results compared to the analytical solution (20)

6. Performance analysis

For the performance analysis, we used 2 different hardware setups:

- **CPU:** Intel Core 2 Duo P7550, 2 x 2.26 GHz; **GPU:** NVIDIA GeForce GT 240M, 48 CUDA cores 1210 MHz, 1 GB DDR3, 128-bit; **OS:** Debian Squeeze 64bit, CUDA Toolkit 3.0.

- **CPU:** AMD FX-8120, 8 x 3.2GHz; **GPU:** NVIDIA GeForce GTX 660Ti, 1344 CUDA cores, 915MHz, 2 GB GDDR5, 192-bit; **OS:** Ubuntu 12.04LTS, CUDA Toolkit 5.5.

The main hardware limitation of both setups was the amount of the GPU memory. In our implementation, the method requires 0.28 GB memory per million particles in 2D and 0.4 GB in 3D. It allows performing simulations of the 3D lid-driven cavity problem using the NVIDIA GTX660Ti (2 GB) up to $Re = 1000$. The numerical efficiency was examined on the basis of simulations of the 2D lid-driven cavity at $Re = 100$. The case is described in detail in Section 5. To analyze the performance of the code, we decided to measure the time of calculation for different numbers of particles $N = 256$, 1024, 4096, 16384, 65536, 262144 and $h/\Delta r = 2$. The time step was constant for all cases $\Delta t = 0.000025|\mathbf{u}_w|/L$, while the simulations ended after obtaining the steady-state solution at $T = 10.0|\mathbf{u}_w|/L$.

Figure 8 presents the computational times as a function of the number of particles in the domain. The results are compared with a similar numerical code designed for performing calculations on a classical CPU.

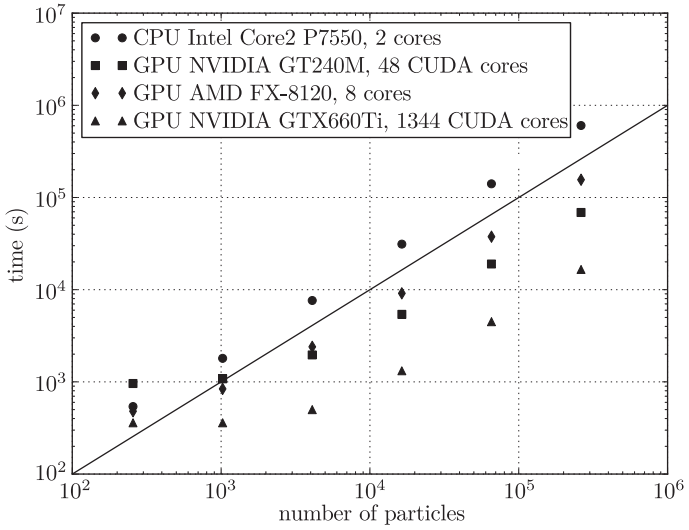

Figure 8. Computational times to obtain the steady-state solution of the lid-driven cavity test case (2D, $Re = 100$)

Figure 9 presents the speed-up of the computational time referred to the CPU calculations on Intel Core2 P7550 (2 cores). Both figures show that the computational performance of the SPH code is much higher using the GPU devices. In the case of medium-class NVIDIA GT660Ti, the time of computation is even 8–10 times smaller comparing to the high-end AMD FX-8120 CPU (8 cores).

In the CUDA programming the efficiency of the code is also dependent on the way how threads are executed within blocks (groups of threads), for details cf. [12]. Different graphic cards have different optimal configurations of threads

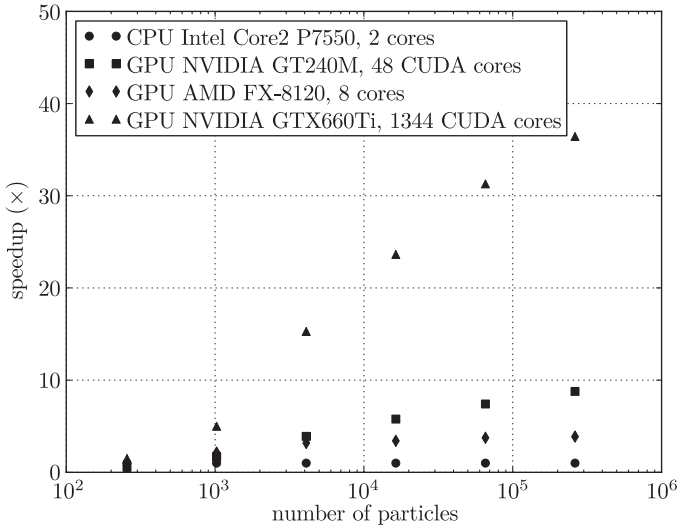

Figure 9. Speedup of computational time refereed to the CPU calculation on Intel Core2 P7550 (2 cores); results obtained for the lid-driven cavity test case (2D, $Re = 100$)

and blocks. Figure 10 presents the computational time of the lid-driven cavity test case for different numbers of threads per block. In the case of the NVIDIA GTX660Ti graphic card the maximal efficiency has been obtained for 128 threads per block, while for NVIDIA GT240M it is 64 threads per block. Another thing that has to be noted is the upper limit of threads per block. Different graphic cards have different maximal number of threads per block. What is more, since we use a shared memory for efficiency reasons, the maximal number of threads per block is limited by the limit of the available shared memory per block. In the case of the NVIDIA GT240M graphic card, the GPU limit is 16384 bytes. Therefore, in our implementation of SPH, the highest possible number of threads per block is 256. In the case of NVIDIA GTX660Ti, the GPU limit of shared memory is 49152 bytes, hence, the highest possible number of threads per block is limited to 1024.

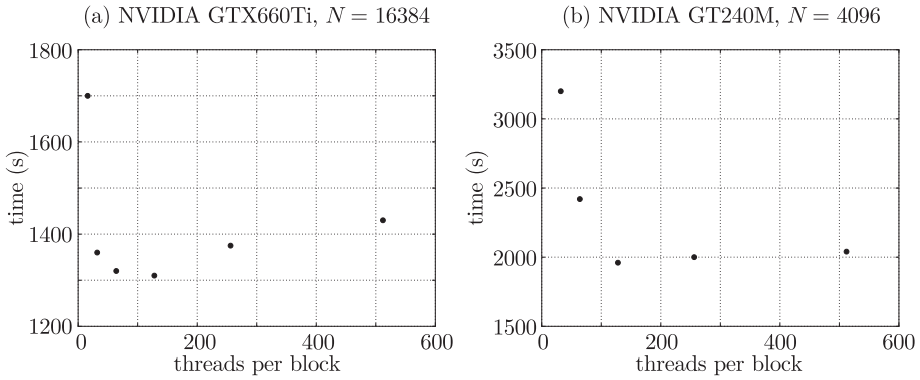

Figure 10. Computational time of the lid-driven cavity test case (2D, $Re = 100$) for different numbers of threads per block; results obtained for two different graphic cards

7. Summary

The SPH method is considered to be a promising approach for modeling complex flow phenomena in engineering. Due to the particle-based nature, no necessity to track the interface position, and no numerical diffusion related to the interface reconstruction, this method appears to be well-suited to handle multiphase flows. What is more, the SPH approach can be easily written in a parallel-manner. Since, in general, the numerical cost of the classical grid-based approaches is smaller, this feature is important and makes the SPH method competitive with grid-based techniques in the field of numerical efficiency. However, in this case, it is necessary to perform the SPH calculations using GPU devices. In the present paper the issues related to the implementation of SPH on GPU have been discussed and the numerical results, including computational times, have been presented. The comparison of numerical efficiency of the GPU implementation run on a mid-class GPU device and a high-class CPU processor has shown higher efficiency of the GPU unit by an order of magnitude. As regards the disadvantages of GPU implementations, the biggest limitation is the amount of the GPU memory. However, it is important to note that this issue may be solved in the nearest future using the heterogeneous Uniform Memory Access (hUMA) by AMD which allows CPUs and GPUs to share the same memory resources (integration of CPU and GPU into a single chip).

Acknowledgements

This research has been funded by Ministry of Science and Higher Education (Poland) via grant Iuventus Plus 0479/IP2/2013/72. The author is indebted to the Polish Science Foundation (FNP) for a research scholarship START 2013.

References

[1] Monaghan J J 2012 *Ann. Rev. Fluid Mech.* **44** 323
[2] Swegle J W, Hicks D L, Attaway S W 1995 *J. Comput. Phys.* **116** 123
[3] Szewc K, Pozorski J, Minier J P 2012 *Int. J. Numer. Methods Eng.* **92** 343
[4] Wendland H 1995 *Adv. Comput. Math.* **4** 389
[5] Cleary P W, Monaghan J J 1999 *J. Comput. Phys.* **148** 227
[6] Szewc K, Tanière A, Pozorski J, Minier J-P 2012 *Int. J. Nonlinear Sci. Numer. Simul.* **13** 383
[7] Szewc K 2013, PhD thesis, Institute of Fluid-Flow Machinery, Polish Academy of Sciences
[8] Hu X Y, Adams N A 2006 *J. Comput. Phys.* **213** 844
[9] Grenier N, Antuono M, Colagrossi A, Le Touzé D, Alessandrini B 2009 *J. Comput. Phys.* **228** 8380
[10] Flekkoy E G, Coveney P V, De Fabritiis G 2000 *Phys. Rev. E* **62** 2140
[11] Cummins S J, Rudman M 1999 *J. Comput. Phys.* **152** 584
[12] Sanders J, Kandrot E 2010 *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley
[13] Cook S 2013 *CUDA programming. A developer's guide to parallel computing with GPUs*, Elsevier
[14] Ghia U, Ghia K N, Shin C T 1982 *J. Comput. Phys.* **48** 387
[15] Lamb H 1932 *Hydrodynamics*, Dover
[16] Morris J P 2000 *Int. J. Numer. Meth. Fluids* **33** 333