Bulletin of the *Transilvania* University of Braşov
Series I: Engineering Sciences • Vol. 6 (55) No. 1 - 2013

GPU ACCELERATED FLUID FLOW COMPUTATIONS USING THE LATICE BOLTZMANN METHOD

C. NIŢĂ[1] L.M. ITU[1] C. SUCIU[1]

Abstract: *We propose a numerical implementation based on a Graphics Processing Unit (GPU) for the acceleration of the execution time of the Lattice Boltzmann Method. The performance analysis is based on three three-dimensional benchmark applications: Poisseuille flow, lid-driven cavity flow and flow in an elbow shaped domain. Three different, recently released GPU cards are considered for the parallel implementation. To correctly evaluate the speed-up potential of the GPUs, both single-core and multi-core CPU based implementations are used. The results indicate that the GTX 680 GPU card leads to the best performance, with a speed-up ranging between 6.7 and 14.35 over the multi-core CPU based implementation, depending on the application and on the grid density.*

Key words: *Lattice Boltzmann Method, parallel computing, computational fluid dynamics, GPU, CUDA.*

1. Introduction

The classic method for studying fluid flow is based on the Navier-Stokes (NS) equations, a system of nonlinear partial differential equations [8]. The numerical solution of the Navier-Stokes equations is one side a difficult task because of the nonlinear terms, and on the other side it is computationally very intensive since a system of algebraic equations, obtained from the Poisson equation, needs to be solved at each time step. Considering additionally the increased accuracy required for engineering or biomedical applications, the flow computation can take several days, even on modern high performance hardware [7].

During the last decades, an alternative approach for studying fluid flow has been proposed: the Lattice Boltzmann Method (LBM) [6]. Unlike the NS equations this method consists of treating the fluid as a system of a large number of particles with known mass. This approach becomes practical by using the kinetic theory of gases, which connects the microscopic physical quantities of the fluid with the macroscopic quantities: pressure, velocity and temperature. The main advantage of this approach is that each grid node can be computed separately based on previously computed values of neighbouring nodes. This aspect enables the efficient parallelization of the LBM [9]. A previous parallel implementation of the LBM gained a speed-up of about 9x on a NVIDIA Tesla C1060 card [5].

[1] Dept. of Automation and Information Technology, *Transilvania* University of Braşov.

With the increasing computational power of Graphics Processing Units (GPU), parallel computing has become available at a relatively small cost. With the advent of CUDA (Compute Unified Device Architecture), several researchers have identified the potential of GPUs to accelerate engineering applications in general, and Computational Fluid Dynamics (CFD) applications in particular to unprecedented levels [4]. Modern GPUs are able to deliver at peak performance over 1 TFLOP, i.e. ten times faster than a multicore CPU.

In this paper we analyze the speed-up potential of the numerical solution of the LBM, using three recently released GPUs, with different architectures. To correctly evaluate the speed-up potential, results are compared against both single-core and multi-core CPU-based implementations.

The paper is organized as follows. In section two we first briefly review the LBM. Then we introduce the numerical implementation, focusing on its parallelization on a GPU. Section three presents detailed results regarding the speed-up obtained with different GPUs and finally, in section four, we draw the conclusions.

2. Methods

2.1. The Lattice Boltzmann Method

For studying the parallel implementation of the LBM, we considered the single relaxation time version of the equation, based on the Bhatnagar-Gross-Krook (BGK) approximation, which assumes that the macroscopic quantities of the fluid are not influenced by most of the molecular collisions (1):

$$\frac{\partial f_i(\mathbf{x},t)}{\partial t} + \mathbf{c}_i \nabla f(\mathbf{x},t) = \frac{1}{\tau}\left(f_i^{eq}(\mathbf{x},t) - f_i(\mathbf{x},t)\right), \quad (1)$$

where: f_i represents the probability distribution function along an axis $\mathbf{c}_i$; τ is a relaxation factor related to the fluid viscosity; $\mathbf{x}$ represents the position and t is the time. The discretization in space and time is performed with finite difference formulas. This is usually done in two steps:

$$f_i(\mathbf{x},t+\Delta t) = f_i(\mathbf{x},t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(\mathbf{x},t) - f_i(\mathbf{x},t)\right),$$

and

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) = f_i(\mathbf{x},t+\Delta t). \quad (2)$$

The first equation is known as the collision step, while the second one represents the streaming step. f^{eq} is called the equilibrium distribution and is given by the following formula:

$$f^{eq} = \omega_i \rho(\mathbf{x},t)\left(1 + \frac{\mathbf{c}_k\mathbf{u}}{c_s} + \frac{1}{2}\frac{\mathbf{c}_k\mathbf{u}^2}{c_s^2} - \frac{1}{2}\frac{\mathbf{u}^2}{c_s^2}\right), \quad (3)$$

where ω_i is a weighting scalar, $c_s = 1/\sqrt{3}$ is the lattice speed of sound and $\mathbf{u}$ is the fluid velocity. $\rho(\mathbf{x},t)$ is a scalar field, commonly called density, which is related to the macroscopic fluid pressure as follows:

$$p(\mathbf{x},t) = \frac{\rho(\mathbf{x},t)}{3}. \quad (4)$$

Once all f_i have been computed, the macroscopic quantities can be determined:

$$\mathbf{u}(\mathbf{x},t) = \frac{1}{\rho(\mathbf{x},t)}\sum_{i=0}^{n}\mathbf{c}_i f_i(\mathbf{x},t), \quad (5)$$

$$\rho(\mathbf{x},t) = \sum_{i=0}^{n} f_i(\mathbf{x},t). \quad (6)$$

The computational domain is similar to a regular grid used for finite difference algorithms. For a more detailed description of the Boltzmann equation and the collision operator we refer the reader to [6].

The current study focuses on three-dimensional flow domains, hence we used the D3Q15 lattice structure, displayed in Figure 1 for a single grid node. A vector $\mathbf{c}_i$ and a scalar weight w_i are defined for each lattice link:

$$\mathbf{c}_i = \begin{cases} (0,0,0) & i = 0 \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & i = 1,..6 \\ (\pm 1,\pm 1,\pm 1) & i = 7,...,14 \end{cases}$$

and the weighting factors are:

$$w_i = \begin{cases} 16/72 & i = 0 \\ 8/72 & i = 1,2,...,5,6 \\ 1/72 & i = 7,8,...,13,14 \end{cases} . \qquad (8)$$

The boundary conditions (inlet, outlet and wall) are crucial for any fluid flow computation. For the LBM, the macroscopic quantities (flow rate/pressure) can not be directly imposed at inlet and outlet.

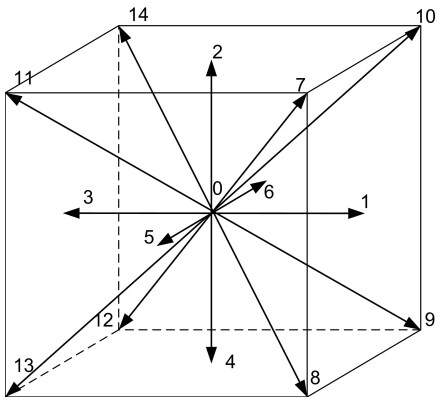


Fig. 1. *The D3Q15 lattice structure, first number in the notation is the space dimension, while the second one is the lattice links number*

Instead, the known values of the macroscopic quantities are used for computing the unknown distribution functions near the boundary. For the inlet and outlet of the domain we used Zou-He [3], [10] boundary conditions with known velocity. The outlet velocity value is set so as to obtain a zero gradient along the boundary normal vector. For the solid walls we used bounce-back boundary conditions based on interpolations, for improving the accuracy when dealing with complex geometries [1]. The solid walls are defined as an isosurface of a scalar field, commonly known as the level-set function. Each point in the field is the signed distance between the grid node and the boundary, therefore the solid wall is located where the scalars are zero.

2.2. GPU based parallel implementation of the Lattice Boltzmann Method

In the following we focus on the parallelization of the LBM, based on a GPU device. The GPU is viewed as a compute device which is able to run a very high number of threads in parallel inside a kernel (a function, written in C language, which is executed on the GPU and launched by the CPU). The GPU contains several streaming multiprocessors, each of them containing several cores. The GPU (usually also called device) contains a certain amount of global memory to/from which the CPU or host thread can write/read, and which is accessible by all multiprocessors. Furthermore, each multiprocessor also contains shared memory and registers which are split between the thread blocks and the threads, which run on the multiprocessor, respectively.

The GPU based implementation of the LBM is performed based on the formulation described in section 2.1. The computations for a single grid node require only the values of the neighbouring nodes from the

previous time step. As a result, each node can be computed at each time step independently from other nodes, and one CUDA thread is used for each grid node. The main difference between the CPU and the GPU implementation of the LBM is the memory arrangement. Regularly, on the CPU, a data structure containing all the required floating-point values for a grid node is defined, and then an array of this data structure is created (the Array Of Structure approach - AOS). This approach is not a viable solution on the GPU because the global memory accesses would not be coalesced and would drastically decrease the performance [11], [12]. Instead of AOS, the Structure Of Arrays (SOA) approach has been considered. E.g. for the fluid velocity, instead of storing all the vector components in one array, three different arrays are used (one for each component). The memory access patterns are described in Figure 2.

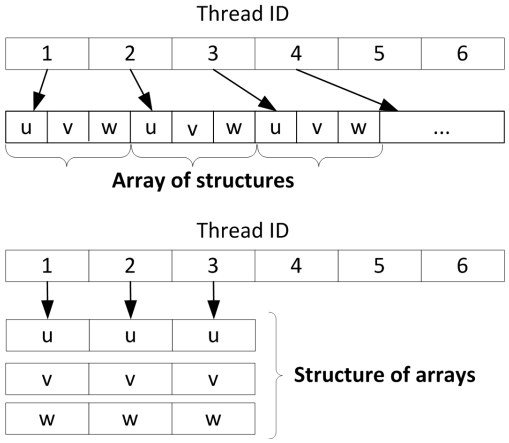


Fig. 2. *Memory access patterns: AOS* (*top*)*, SOA* (*bottom*)

The LBM requires a total of 33 arrays, 15 for the density functions, another 15 for swapping the new density functions with the old ones after the streaming step, three for the velocity, one for the density and another one for the level-set function used for the solid boundaries. Note that we used double precision floating-point numbers since single precision does not meet the accuracy requirements. Double precision floating-point operations are only available on GPU devices with compute capability 1.3 or greater. Furthermore, all arrays are one-dimensional. Mapping the three-dimensional coordinates to a global index inside the array is performed as follows:

$$i_g = i \cdot N_y N_z + j \cdot N_z + k \,. \tag{9}$$

And the other way around:

$$\begin{aligned} i &= \frac{i_g}{N_y N_z}, \\ j &= \frac{i_g - i \cdot N_y N_z}{N_z}, \\ k &= i_g - i \cdot N_y N_z - j \cdot N_z, \end{aligned} \tag{10}$$

where: i, j and k are the node coordinates in the three-dimensional grid. Note that these values are approximated with the floor function, N_x, N_y and N_z are the grid sizes in each direction and i_g is the global index of the node in the one-dimensional array. Both (9) and (10) are used at the streaming step for finding the global index of the neighbouring nodes.

The workflow of the GPU based LBM implementation is displayed in Figure 3. The computations required for a time step are entirely performed on the GPU. Therefore, host-device memory copy operations are only required when storing intermediate results (e.g. for observing transient or unsteady flows).

A total of four kernels are called at each iteration. The kernel which computes the macroscopic quantities (velocity and density) iterates through the 15 probability distribution functions, and applies (5) and (6). All computations are performed only for the fluid nodes. The results are stored in the global memory. The computations for the collision step are performed similarly:

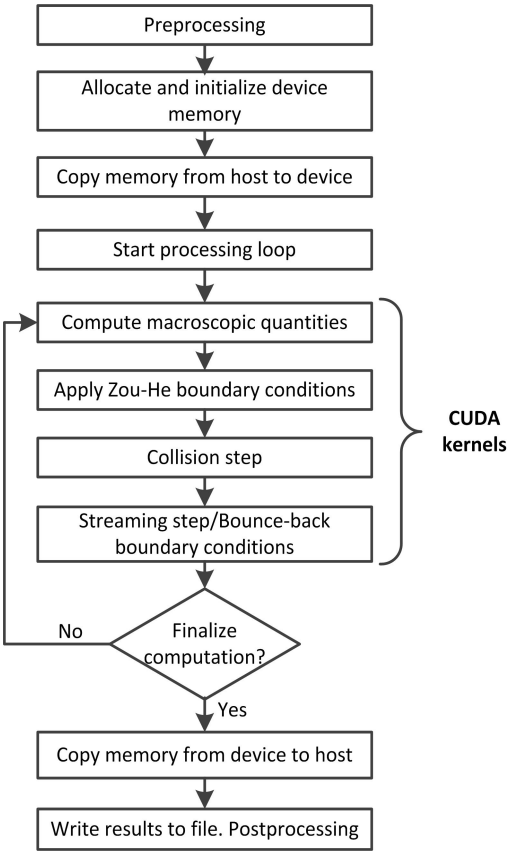


Fig. 3. *Computation workflow*

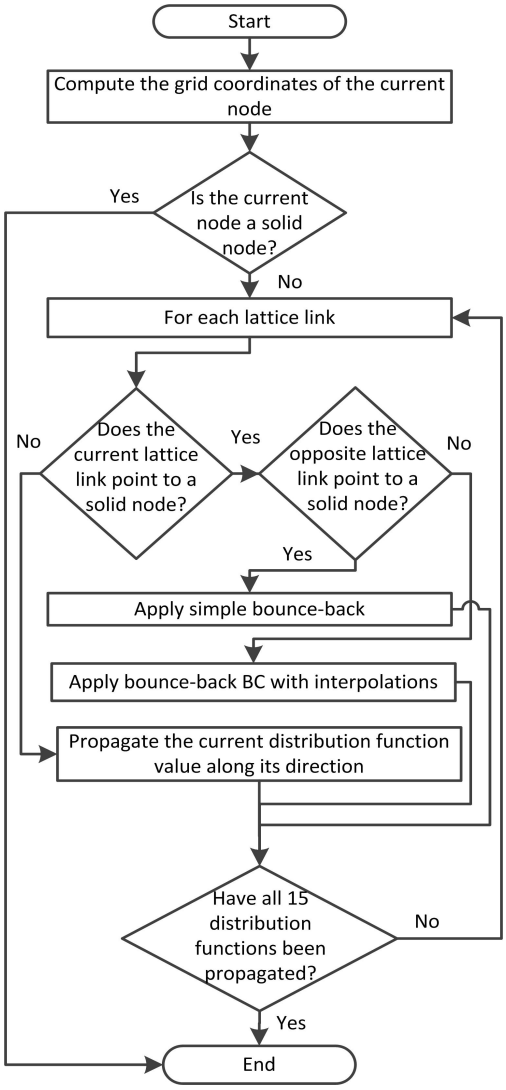


Fig. 4. *The workflow of the streaming step*

first the equilibrium distribution function is computed using (3) and then the new probability distribution functions are determined based on (2). Since the grid nodes located at the boundary require a different treatment than the other nodes, the streaming step is more complex. This leads to different code execution paths and therefore to reduced parallelism. However, since very few grid nodes reside next to the boundary, this aspect is not crucial for the overall performance. The workflow of the streaming step is described in Figure 4.

3. Results and Discussion

To compare the performance of the CPU based implementation of the LBM with the GPU based implementation, we considered three different NVIDIA GPU cards: GeForce GTX 460, GeForce GTX 650 and GeForce GTX 680 (the first one is based on the Fermi architecture, while the other two are based on the Kepler architecture). The CPU based implementation was run on an eight-core i7 processor using both single and multi-threaded code. Parallelisation of the CPU code was performed using OpenMP.

Three different benchmark applications were considered for performance comparison: Poisseuille flow, lid-driven cavity flow and flow in an elbow shaped domain (Figures 5 and 6). Different grid resolutions were

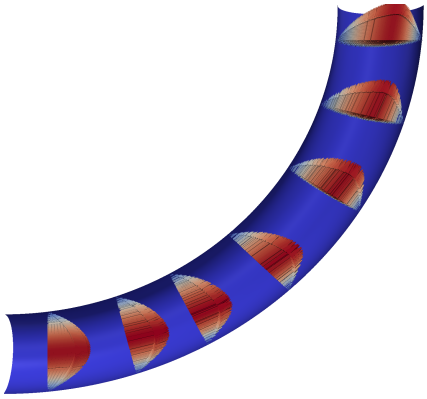
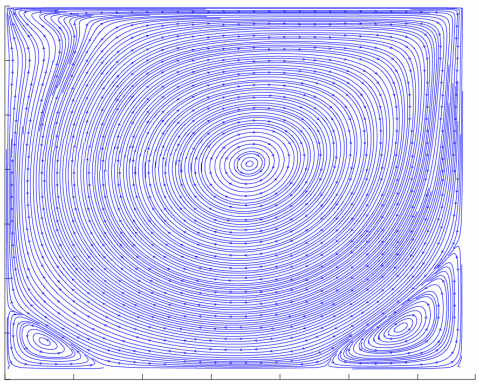


Fig. 5. *Flow through an elbow shaped domain*

considered, leading to grids with 0.25 to 4 million nodes. The grid configuration is different for each of these cases hence the execution time and GPU-CPU speed-up is also different. Table 1 displays the execution times for all test cases. The computations on the grid with 4 million nodes could only be run on the GTX 680 card, the other two having not enough memory. Note that these execution times correspond to only one computation step. The total number of computation steps to obtain convergence strongly depends on the grid resolution, i.e. the time needed by the pressure wave to propagate from one end to the other which is given by the lattice speed of sound. For the Poisseuille flow computations on the 50x50x200 grid, 5000 steps were required for reaching steady state and twice as much for the 100x100x400 grid. A similar number of steps is required for the computations in the elbow shaped domain.

However, for the lid driven cavity flow steady state is reached after a much higher number of steps. In this case the fluid movement is not given by the pressure drop but by the moving lid which causes the fluid to move along with it because of the shear stress. Velocity propagation due to shear stress is much slower than in a pressure-driven flow, hence a much larger number of time steps are required (50000 steps). Figure 6 top displays the streamlines of a lid-driven cavity flow.

The performance improvements are significant and demonstrate that a GPU based implementation of the LBM is superior to a multi-core CPU based implementation. The best performance has been obtained for the GTX 680, as the results displayed in bold in Table 1 show. For grids with more than 0.5 million nodes, the speed-up is around 14x on the GTX 680 and higher than 10x for smaller grids. For the elbow case, since about 83% of the grid nodes are solid nodes and don't require any computation, the speed-up is of only 7x for a grid with 0.5 million nodes and 10x for 2 million nodes. Figure 7 displays a comparison of the execution times for all three employed GPUs and the multi-threaded CPU code. Note that the performance of the GTX 650 card is around 2x lower than the GTX 460, an aspect



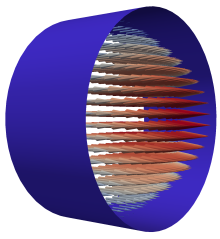


Fig. 6. *Lid-driven cavity flow at Re = 1000* (*left*) *and Poisseuille flow* (*top*)

Measurements of the execution times for one computation step Table 1

Bench-mark case	Grid resolution	Single-threaded CPU code [ms]	Multi-threaded CPU code [ms]	GeForce GTX 680		GeForce GTX 650		GeForce GTX 460	
				Time [ms]	Speed-Up	Time [ms]	Speed-Up	Time [ms]	Speed-Up
Poisseuille flow	100x100x400	3924.8	608.38	**42.4**	**14.35**	-	-	-	-
	50x50x200	484.3	81.39	**5.7**	**14.28**	17.5	4.65	9.6	8.48
	25x25x100	61.01	11.24	**1**	**11.24**	3	3.75	1.7	6.61
Lid-driven cavity flow	100x100x100	977.94	152.48	**11**	**13.86**	34.2	4.46	18.8	8.11
	50x50x50	120.81	20.34	**1.8**	**11.30**	5	4.07	2.9	7.01
	25x25x25	15.09	3.35	**0.5**	**6.70**	1.1	3.05	0.9	3.72
Elbow	200x200x50	1956.12	91.02	**0.5**	**10.71**	1.1	3.13	0.9	5.03
	100x100x50	242.46	12.0	**8.5**	**7.06**	29.1	2.61	18.1	4.14

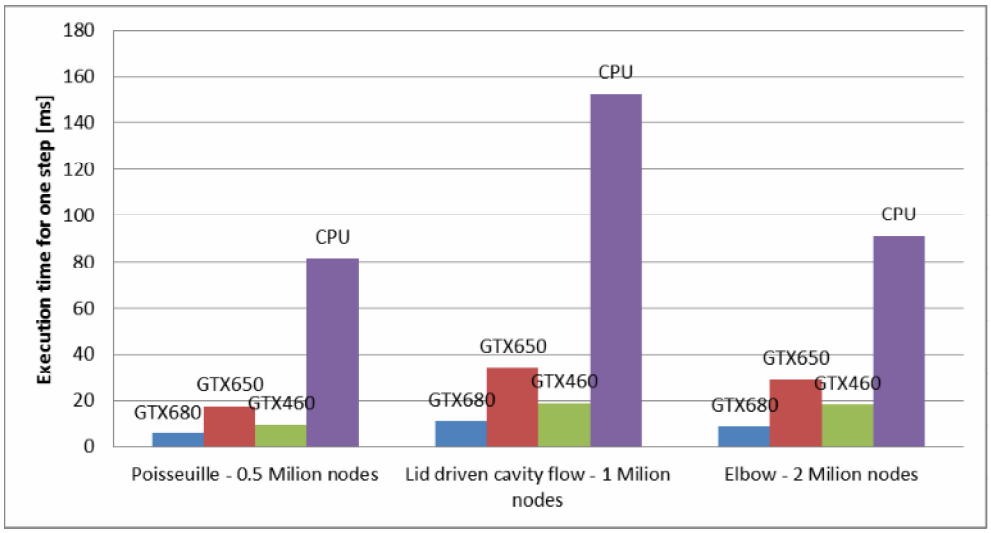


Fig. 7. *Comparison of the execution times*

which confirms the concerns raised for the first GPUs of the Kepler architecture, which stated that the performance is in fact lower than for the previously released cards of the 400 and 500 GeForce series. According to NVIDIA, the focus was directed towards lower power consumption for the GPUs from the GeForce 600 series.

Although the parallel implementation of LBM leads to a significant performance increase, the algorithm does not fully benefit from the GPU power mainly because of the global memory accesses, which is a commonly encountered bottleneck of GPU based computational workflows. This drawback is prominent at the streaming step where the majority of global memory accesses are required. The streaming step occupies around 50% of the execution time of a computation step. Another limiting factor is given by the boundary nodes, which require a special treatment.

4. Conclusions

In this paper, we introduced a GPU based parallel implementation of the Lattice Boltzmann Method. We focused on optimizing the global memory access by using the SOA approach instead of the AOS approach and 1-D arrays. Although the GPU is not used at its full capacity the

performance improvement was significant, the GTX680 leading to a speed-up of over 14x for grids with more than 0.5 million nodes and of 10x for smaller grids, compared to a multi-core CPU-based implementation.

The acceleration of the execution time of the LBM is an important aspect, especially since the LBM has been used increasingly for blood flow computations [2]. When blood flow is modelled in patient-specific geometries in a clinical setting, results are required in a timely manner not only to potentially treat the patient faster, but also to perform computations for more patients in a certain amount of time.

Future work will be directed towards further optimization of the global memory accesses. Moreover, a multi-GPU based implementation will be considered to further increase the parallelism.

Acknowledgements

This work is supported by the program Partnerships in Priority Domains (PN II), financed by ANCS, CNDI - UEFISCDI, under the project nr. 130/2012.

References

1. Bouzidi, M., Firdaouss, M., Lallemand, P.: *Momentum Transfer of a Boltzmann-Lattice Fluid with Boundaries*. In: Physics of Fluids **13** (2001) No. 11, p. 452-3459.
2. Golbert, D.R., Blanco, P.J., Clausse, A., Feijóo, R.A.: *Tuning a Lattice-Boltzmann Model for Applications in Computational Hemodynamics*. In: Medical Engineering & Physics **34** (2012) No. 3, p. 339-349.
3. Ho, C.H., Chang, C., Lin, K.H., Lin, C.A: *Consistent Boundary Conditions for 2D and 3D Lattice Boltzmann Simulations*. In: Computer Modeling in Engineering and Sciences **44** (2009) No. 2, p. 137-155.
4. Kirk, D., Hwu, W.M.: *Programming Massively Parallel Processors: A Hands-on Approach*. London. Elsevier, 2010.
5. Riegel, E., Indinger, T, Adams, N.A.: *Implementation of a Lattice-Boltzmann Method for Numerical Fluid Mechanics using the nVIDIA CUDA Technology*. In: Computer Science-Research and Development **23** (2009) No. 3, p. 241-247.
6. Succi, S.: *The Lattice Boltzmann Equation - For Fluid Dynamics and Beyond*. New York. Oxford University Press, 2001.
7. Taylor, C.A., Steinman, D.A.: *Image-based Modeling of Blood Flow and Vessel Wall Dynamics: Applications, Methods and Future Directions*. In: Annals of Biomedical Engineering **38** (2010) No. 3, p. 1188-1203.
8. Temam, R.: *Navier-Stokes Equations - Theory and Numerical Analysis*. Amsterdam. North Holland Publishing, 1977.
9. Tölke, J.: *Implementation of a Lattice Boltzmann Kernel using the Compute Unified Device Architecture developed by nVIDIA*. In: Computing and Visualization in Science **13** (2010) No. 1, p. 29-39.
10. Zou, Q., He, X.: *On Pressure and Velocity Boundary Conditions for the Lattice Boltzmann BGK Model*. In: Physics of Fluids **9** (1997) No. 6, p. 1591-1598.
11. NVIDIA Corporation: *CUDA, Compute Unified Device Architecture Programming Guide v5.0* (2013).
12. NVIDIA Corporation: *CUDA, Compute Unified Device Architecture Best Practices Guide v5.0* (2013).