GPU Accelerated Blood Flow Computation using the Lattice Boltzmann Method

Cosmin Niță, Lucian Mihai Itu, Constantin Suciu
Department of Automation
Transilvania University of Braşov
Braşov, Romania

Constantin Suciu
Corporate Technology
Siemens
Braşov, Romania

***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 (LBM). The study focuses on the application of the LBM for patient-specific blood flow computations, and hence, to obtain higher accuracy, double precision computations are employed. The LBM specific operations are grouped into two kernels, whereas only one of them uses information from neighboring nodes. Since for blood flow computations regularly only 1/5 or less of the nodes represent fluid nodes, an indirect addressing scheme is used to reduce the memory requirements. Three GPU cards are evaluated with different 3D benchmark applications (Poisseuille flow, lid-driven cavity flow and flow in an elbow shaped domain) and the best performing card is used to compute blood flow in a patient-specific aorta geometry with coarctation. The speed-up over a multi-threaded CPU code is of 19.42x. The comparison with a basic GPU based LBM implementation demonstrates the importance of the optimization activities.**

Keywords*— *Lattice Boltzmann Method, parallel computing, GPU, CUDA, coarctation of the aorta

I. INTRODUCTION

In recent years, there has been considerable focus on computational approaches for modeling the flow of blood in the human cardiovascular system. When used in conjunction with patient-specific anatomical models extracted from medical images, such techniques provide important insights into the structure and function of the cardiovascular system [1].

The Lattice Boltzmann Method (LBM) has been introduced in the 80's, and has developed into an alternative powerful numerical solver for the Navier-Stokes (NS) equations for modeling fluid flow. Specifically, LBM has been used consistently in the last years in several blood flow applications (e.g. coronaries [2], aneurysms [3], abdominal aorta [4]). The LBM is a mesoscopic particle based method, which has its origin in the Lattice Gas Automata. It uses a simplified kinetic model of the essential physics of microscopic processes, such that the macroscopic properties of the system are governed by a certain set of equations. The equation of LBM is hyperbolic, and can be solved explicitly and efficiently on parallel computers [5]. 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 Computational Fluid Dynamics (CFD) applications to unprecedented levels [6].

Due to the high computational requirements, there has been a lot of interest in exploring high performance computing techniques for speeding up the LBM algorithms. Efficient CUDA based implementations of the 3D LBM have been proposed previously in the literature [7-10], which were optimized for specific applications. Tölke et al. [10] obtained a speed-up of around 100x over a sequential implementation on the Intel Xeon CPU for the flow around a moving sphere. Obrecht et al. [9] studied the flow in an urban environment and obtained for a multi-GPU implementation a speed-up of 28x compared to a multi-threaded CPU based implementation. All these researches focused on single precision computations. With the introduction of the Fermi and the Kepler architecture, the performance of double precision computations on NVIDIA GPU cards has increased substantially.

In this paper we introduce a parallel implementation of the LBM designed for blood flow computations. To meet the high accuracy requirements of blood flow applications, computations are performed with double precision. Three recently released GPUs have been considered and, to correctly evaluate the speed-up potential, results are compared against both single-core and multi-core CPU-based implementations. The best performing GPU card is first determined using three popular benchmarking applications, and then it is used for computing blood flow in a patient-specific aorta geometry with coarctation (CoA), containing the descending aorta and the supra-aortic branches. CoA is a congenital cardiac defect usually consisting of a discrete shelf-like narrowing of the aortic media into the lumen of the aorta, occurring in 5 to 8% of all patients with congenital heart disease [11]. The narrowing can lead to a significant pressure drop, which affects the health of the patient. Both the importance and the potential of CFD based approaches for non-invasive diagnosis of CoA patients have been recently emphasized in a challenge [12], where the LBM produced good results.

The paper is organized as follows. In section two we first briefly introduce the LBM used herein. Then we introduce the numerical implementation, focusing on its optimized parallelization on a GPU. Section three first presents detailed results for the speed-up obtained with different GPUs for the benchmarking applications, and then it displays the results obtained with the best performing GPU card for the patient

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

specific CoA geometry. Finally, in section four, we draw the conclusions.

II. METHODS

A. 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:

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

where f_i represents the probability distribution function along an axis $\boldsymbol{c}_i$, τ is a relaxation factor related to the fluid viscosity, $\boldsymbol{x}$ 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:

$$f_i(\boldsymbol{x}, t+\Delta t) = f_i(\boldsymbol{x},t) + \frac{\Delta t}{\tau}(f_i^{eq}(\boldsymbol{x},t) - f_i(\boldsymbol{x},t)), \quad (2a)$$

and

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

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:

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

where ω_i is a weighting scalar, c_s is the lattice speed of sound, $\boldsymbol{c}_k$ is the direction vector, and $\boldsymbol{u}$ is the fluid velocity. $\rho(\boldsymbol{x}, t)$ is a scalar field, commonly called density, which is related to the macroscopic fluid pressure as follows:

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

Once all f_i have been computed, the macroscopic quantities (velocity and density) can be determined:

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

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

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 [5]. The current study focuses on 3D flow domains: we used the D3Q15 lattice structure, displayed in fig. 1 for a single grid node. The weighting factors are: $\omega_i = 16/72$ for $i = 0$, $\omega_i = 8/72$ for $i = 1\ldots6$, and $\omega_i = 1/72$ for $i = 7\ldots14$.

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. 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 [13] boundary conditions with known velocity. For the outlet we used homogeneous Neumann boundary condition. The arterial geometry has complex boundaries in patient-specific blood flow computations, and hence, for improving the accuracy of the results, we used advanced bounce-back boundary conditions based on interpolations [14]. The solid walls are defined as an isosurface of a scalar field, commonly known as the level-set function.

B. GPU based parallel implementation of the Lattice Boltzmann Method

In the following we focus on the GPU based parallelization of the above described LBM. 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 contains a certain amount of global memory to/from which the CPU 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 LBM is both computationally expensive and memory demanding [15], but its explicit nature and the data locality (the computations for a single grid node require only the values of the neighboring nodes) make it ideal for parallel implementations. Each node can be computed at each time step independently from other nodes. A first important 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 Structures 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 [16]. Instead of AOS, the Structure Of Arrays (SOA) approach has been considered [15]: a different array is allocated for each variable of a node, leading to a total of 35 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

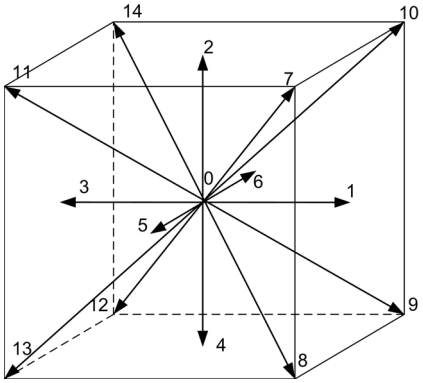

Fig. 1. The D3Q15 lattice structure, first number in the notation is the space dimension, while the second one is the lattice links number.

density and one for the level-set function. The memory access patterns for the AOS and SOA approaches are displayed in fig. 2 for the three velocity components. The workflow of the GPU-based LBM implementation is displayed in fig. 3. All computations are performed on the GPU. Therefore, host-device memory copy operations are only required when storing intermediate (transient or unsteady flows) or final results (steady flows).

Two different kernels have been defined and are called at each iteration. The operations in (2) – (6) have been associated to the two kernels based on the necessity of accessing information from the neighboring nodes. Kernel 1 first computes the macroscopic quantities (velocity and density), based on (5) and (6), by iterating through the 15 probability distribution functions. Then it applies the Zou-He boundary conditions at the inlet of the domain and it performs the collision step: first the equilibrium distribution function is computed using (3) and then the new probability distribution functions are determined based on (2). The second kernel focuses on the streaming step, the interpolated bounce-back boundary condition and the outlet boundary condition. All these operations require information from the neighboring nodes. The operations of the second kernel are more complex since the grid nodes located at the boundary require a different treatment than the other nodes. This leads to different code execution paths and therefore to reduced parallelism. However, since relatively few grid nodes reside next to the boundary, this aspect is not crucial for the overall performance. The workflow of the streaming step is displayed in fig. 4 (for simplicity, the treatment of the nodes of the outlet boundary is not displayed). One can see that, if a node is surrounded in opposite directions by solid nodes, the simple bounce back rule is applied instead of the interpolated bounce back rule, which would lead to numerical divergence. This case is encountered relatively often in geometries with complex boundaries, especially around sharp edges. For both kernels, one CUDA thread is mapped to one node and since all arrays are one-dimensional, also the execution configuration of the kernels is one-dimensional, both at block and at grid level.

Due to the high accuracy requirements of blood flow computations, and unlike previous researches, all computations were performed with double precision. Because the arrays and the execution configuration are one-dimensional, it is necessary

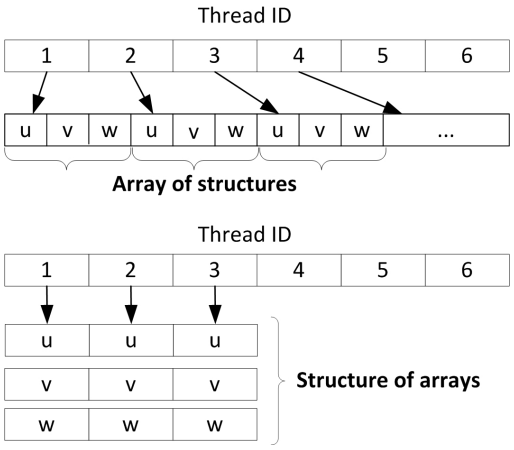

Fig. 2. Memory access patterns: Array of Structures (top), Structure of Arrays (bottom).

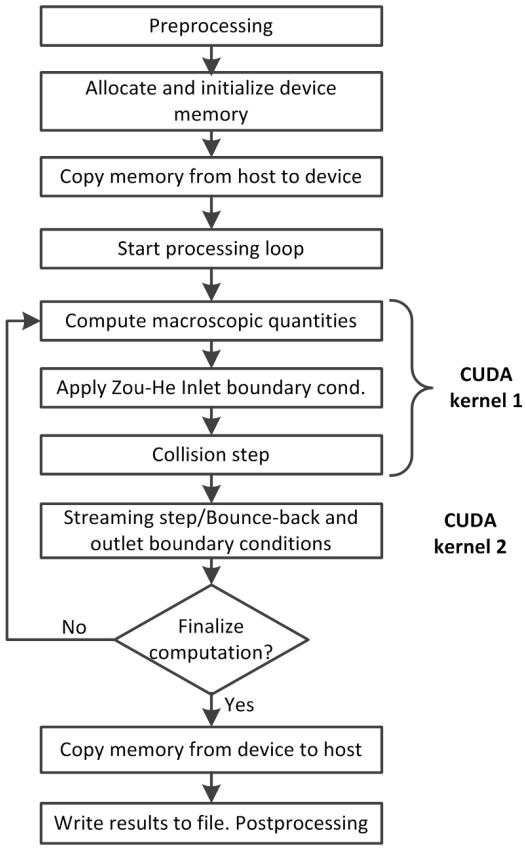

Fig. 3. LBM workflow.

to map the three-dimensional coordinates inside the grid to a global index used to access the data from the arrays:

$$i_g = i \cdot N_y N_z + j \cdot N_z + k \,. \tag{7}$$

$$\begin{aligned} i &= \frac{i_g}{N_y N_z}, \\ j &= \frac{i_g - i \cdot N_y N_z}{N_z}, \\ k &= i_g - i \cdot N_y N_z - j \cdot N_z. \end{aligned} \tag{8}$$

where i, j and k are the node coordinates in the 3D LBM 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. Equations (7) and (8) are used inside the second kernel for finding the global index of the neighbouring nodes.

The LBM is usually applied for a rectangular grid. For blood flow computations, the rectangular grid is chosen so as to include the arterial geometry of interest. In this case though, the fluid nodes represent only 1/5 or less of the total number of nodes. Hence, if the nature of the nodes (fluid/solid) is not taken into account, around 80% of the allocated memory is not used and around 80% of the threads do not perform any computations. To avoid this problem, we used an indirect addressing scheme, displayed in fig. 5. Memory is only

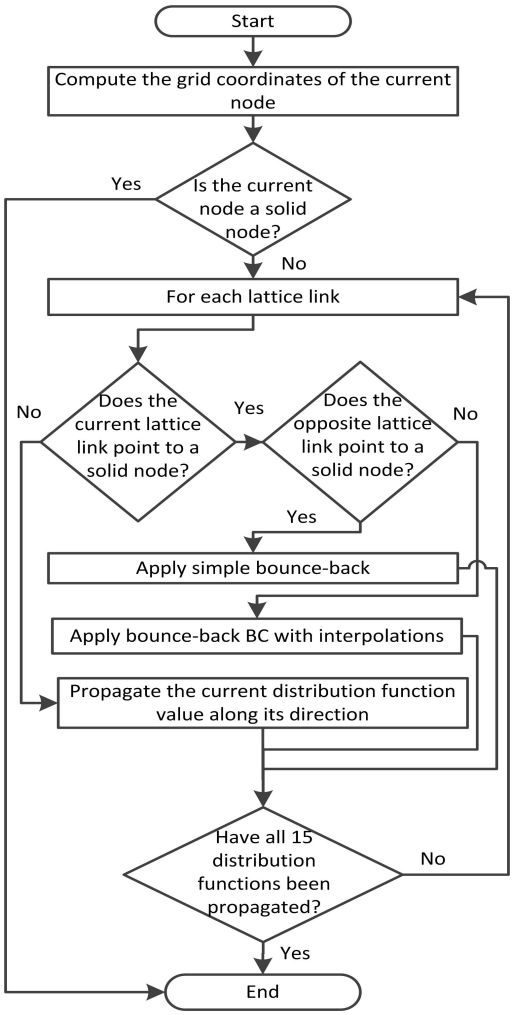

Fig. 4. The workflow of the second kernel in fig. 3.

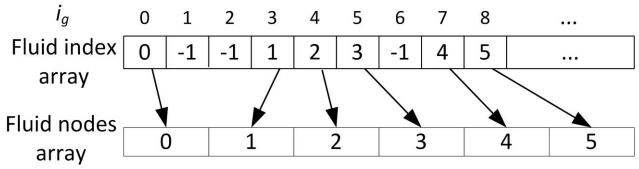

Fig. 5. Indirect addressing.

allocated for the fluid nodes and an additional array (called fluid index array) is introduced for mapping the global index determined with (7) to the fluid nodes arrays (negative values in the fluid index array correspond to solid nodes). The content of the fluid index array is determined in the preprocessing stage on the CPU and is required only during the streaming step. Since for the operations performed inside the first kernel in fig. 3 no information from the neighboring nodes is required, the execution configuration of the first kernel is set up so as to generate a number of threads equal to the number of fluid nodes. For the second kernel on the other side, the number of threads in the execution configuration is set equal to the total number of nodes, to avoid the necessity of a search operation in the fluid index array.

III. RESULTS

To compare the performance of the CPU based implementation of the LBM with the GPU based implementation for double precision computations, 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. Parallelization of the CPU code was performed using OpenMP.

Three different 3D benchmark applications were first considered for determining the best performing GPU card: Poisseuille flow, lid-driven cavity flow and flow in an elbow shaped domain. Different grid resolutions were considered and table I displays the execution times for all test cases, corresponding to one computation step.

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 is obtained for the GTX 680 (see table I). The speed-up is computed based on the multi-threaded CPU code. The speed-up compared to the single-threaded CPU code varies between 150x and 290x. Note that the performance of the GTX 650 card is on average around 2x lower than of the GTX 460. This confirms the concerns raised for the first GPUs of the Kepler architecture, the performance of which are in fact lower than for the previously released cards of the 400 and 500 GeForce series (with the advantage of lower power consumption).

Once the GTX680 was determined as best performing GPU card for double-precision 3D computations, we used it to compute blood flow in a patient-specific aorta model with coarctation, which was recently used in a CFD challenge [12]. To obtain the correspondence between the lattice units and the physical units, we used the method described in [17]. The computations were initialized with the equilibrium distribution function, and for the current research activity we focused on steady-state computations, i.e. we imposed the average value of the flow rate profile specified in the challenge. The grid size was set to 92x156x428 (6142656 nodes), of which only 518969 represented fluid nodes (less than 10%). 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, an aspect which is given by the lattice speed of sound. Fig. 6 displays the computation results obtained after 10000 time steps (the converged solution). Following the idea in [18], namely that lower occupancy leads to better performance, we tested different execution configurations. The execution times obtained for different thread block configurations, for the entire computation, are displayed in table II alongside the execution time for the multi-threaded CPU code. As has been reported previously [15], execution configurations with fewer threads per block lead to better performance. The best performing execution configuration is with 128 threads per block and the speed-up compared to the execution time of the multi-threaded CPU implementation is of **19.42x**.

TABLE I. EXECUTION TIMES OF BENCHMARKING APPLICATIONS FOR ONE COMPUTATION STEP FOR DIFFERENT GRID CONFIGURATIONS.

Benchmark 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	13.7	**44.41**	45.30	13.43	21.00	28.97
	50x50x200	484.3	81.39	1.9	**42.84**	6.00	13.57	3.00	27.13
	25x25x100	61.01	11.24	0.30	**37.47**	0.80	14.05	0.50	22.48
Lid-driven cavity flow	100x100x100	977.94	152.48	6.40	**23.83**	21.40	7.13	9.20	16.57
	50x50x50	120.81	20.34	0.80	**25.43**	2.70	7.53	1.20	16.95
	25x25x25	15.09	3.35	0.10	**33.50**	0.40	8.38	0.30	11.17
Elbow	200x200x50	1956.12	91.02	2.50	**36.41**	8.60	10.58	4.40	20.69
	100x100x50	242.46	12.0	0.90	**13.33**	2.80	4.29	0.70	17.14

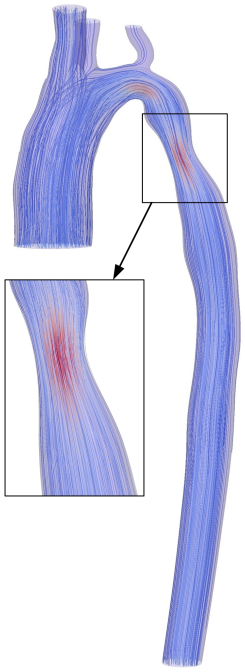

Fig. 6. Computation result (streamlines) for the patient-specific coarctation geometry.

TABLE II. COMPARISON OF EXECUTION TIMES FOR DIFFERENT EXECUTION CONFIGURATIONS

Configuration	Execution time [s]
GPU - 64 threads/block	37.160
GPU - 128 threads/block	34.654
GPU - 256 threads/block	35.743
GPU - 512 threads/block	35.825
GPU - 1024 threads/block	39.989
CPU - multithreaded	673.028

The implementation and optimization aspects described in the previous section were designed specifically for blood flow computations. To evaluate the impact of these activities we also performed the flow computations in the same model with a basic version of the LBM GPU implementation. The basic LBM GPU version did not use indirect addressing (memory

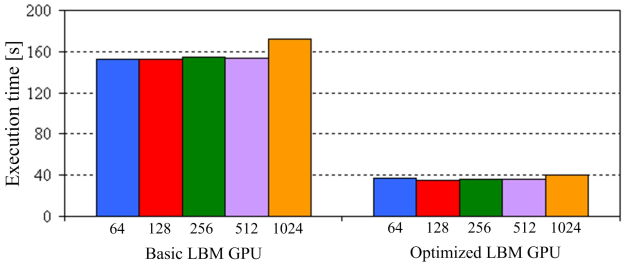

Fig. 7. Comparison of basic vs optimized LBM GPU implementation.

was allocated for all nodes, including the solid nodes), used four kernels for the operations of the LBM at each iteration, and executed all kernels with a total number of threads equal to the total number of nodes. The results are displayed in fig. 7 for different thread block configurations and show that the optimization activities are crucial for the speed-up (with the basic LBM GPU version, the speed-up is of only 4.41x compared to the multi-threaded CPU code). The speed-up of the optimized LBM GPU version compared to the basic LBM GPU version is of 4.40x.

IV. DISCUSSION AND CONCLUSIONS

In this paper, we introduced a GPU-based parallel implementation of the Lattice Boltzmann Method, optimized for patient-specific blood flow computations. Double precision computations were employed for higher accuracy and three different NVIDIA GPU cards were considered. Based on three 3D benchmarking applications, the GTX680 card was determined as best performing GPU and was subsequently used to compute blood flow in a aorta geometry with coarctation.

To our knowledge, this is the first work to evaluate the potential of Kepler architecture GPU cards for accelerating the execution of the LBM. Moreover, it is the first paper to consider double precision computations for higher accuracy. A detailed comparison with previous implementations [7-10] is difficult to perform since generally the implementations are optimized for specific activities and different GPUs have been used in different studies. However, the overall results obtained herein are remarkable: the speed-up over a single-threaded CPU implementation varies between 150x and 290x, whereas previously a speed-up of 100x was reported [10]. The speed-up of the CoA geometry blood flow computation was of 19.42x

compared to a multi-threaded CPU implementation, whereas previously a speed-up of 28x was reported, but for a multi-GPU and not a single GPU implementation [9].

The optimization activities were designed for patient-specific blood flow computations in general (not in particular for the coarctation geometry), where the ratio of fluid nodes to total number of nodes is usually around 1/5 or less. Hence we used an indirect adressing scheme and allocated memory only for the fluid nodes. Furthermore, the operations were grouped into two kernels: the first one performs operations for which information from neighboring nodes is not required, while the second one uses information from neighboring nodes. This way the number of kernels is reduced, and it was possible to use an execution configuration with reduced number of threads for the operations for which information from the neighboring nodes is not required. As proposed in the CFD challenge [12], we only considered rigid wall computations. If elastic arterial walls are considered, then the fluid index array in fig. 5 has to be recomputed at each time step since the classification of nodes into fluid and solid nodes changes over time.

All LBM based results reported for [12] were obtained for CPU based implementations. Although the LBM is faster than the classic CFD approach, based on the Navier-Stokes equations, the acceleration of the execution time remains a crucial task for several reasons. First of all, 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. Furthermore, when performing patient-specific computations, it is necessary to match certain patient-specific characteristics, like pressure or flow rates. Hence, the parameters of the model need be tuned, and the computation needs to be run repeatedly for the same geometry, thus increasing the total execution time for a single patient [19].

Several future work activities have been identified. From a computational point of view, the global memory accesses of the second kernel can be further optimized, and a multi-GPU based implementation will be considered for further decreasing the execution time. From a modeling point of view, for more severe coarctations than the one displayed in fig. 6, the Reynolds number increases considerably and a Smagorinsky sub-grid model needs to be employed [9].

ACKNOWLEDGMENT

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] C.A. Taylor, and D.A. Steinman, “Image-based modeling of blood flow and vessel wall dynamics: applications, methods and future directions,” Annals of Biomedical Engineering, vol. 38, pp. 1188-1203, 2010.

[2] S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, Mitsouras D, et al., “Hydrokinetic approach to large-scale cardiovascular blood flow,” Computer Physics Communications, vol. 181, pp. 462-72, 2010.

[3] J. Bernsdorf, and D. Wang, “Non-Newtonian blood flow simulation in cerebral aneurysms,” Computers & Mathematics with Applications, vol. 58 pp. 1024-1029, 2009.

[4] A.M. Artoli, A.G. Hoekstra, and P.M.A. Sloot, “Mesoscopic simulations of systolic flow in the human abdominal aorta”, Journal of Biomechanics, vol. 39, pp. 873-874, 2006.

[5] S. Succi, The Lattice Boltzmann Equation - For Fluid Dynamics and Beyond. New York: Oxford University Press, 2001.

[6] D. Kirk, and W.M. Hwu, Programming Massively Parallel Processors: A Hands-on Approach. London: Elsevier, 2010.

[7] P. Bailey, J. Myre, S.D.C. Walsh, D.J. Lilja, and M.O. Saar, “Accelerating lattice Boltzmann fluid flow simulations using graphics processors,” IEEE International Conference on Parallel Processing, Vienna, Austria, pp. 550-557, Sept. 2009.

[8] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras, “A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries,” Concurrency Computation: Practice & Experience, vol. 22, pp. 1-14, 2010.

[9] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux, “Towards urban-scale flow simulations using the Lattice Boltzmann Method,” Building Simulation Conference, Sydney, Australia, pp. 933-940, Nov. 2011.

[10] J. Tölke, and M. Krafczyk, “TeraFLOP computing on a desktop PC with GPUs for 3D CFD,” International Journal of Computational Fluid Dynamics, vol. 22, pp. 443-456, 2008.

[11] R.E. Ringel, and K. Jenkins, “Coarctation of the aorta stent trial (coast)”, 2007, http://clinicaltrials.gov/ct2/show/NCT00552812.

[12] ***, CFD Challenge: Simulation of Hemodynamics in a Patient-Specific Aortic Coarctation Model, http://www.vascularmodel.org/miccai2012/.

[13] Q. Zou, and X. He, “On pressure and velocity boundary conditions for the Lattice Boltzmann BGK model,” Physics of Fluids, vol. 9, pp. 1591-1598, 1997.

[14] M. Bouzidi, M. Firdaouss, and P. Lallemand, “Momentum transfer of a Boltzmann-Lattice fluid with boundaries,” Physics of Fluids, vol. 13, pp. 452-3459, 2001.

[15] M. Astorino, J. Becerra Sagredo, and A. Quarteroni, “A modular lattice Boltzmann solver for GPU computing processors”, SeMA journal, vol. 59, pp. 53-78, 2012.

[16] NVIDIA Corporation: CUDA, Compute Unified Device Architecture Best Practices Guide v5.0 (2013).

[17] J. Latt, “Hydrodynamic limit of lattice Boltzmann equations”, PhD Thesis, Universite de Geneve, Geneve, Switzerland, 2007.

[18] V. Volkov, “Better performance at lower occupancy,” GPU Technology Conference, San Jose, USA, 2010.

[19] D.R.Golbert, P.J. Blanco, A. Clausse, and R.A. Feijóo, “Tuning a Lattice-Boltzmann model for applications in computational hemodynamics,” Medical Engineering & Physics, vol. 34, pp. 339-349, 2012.