ISSN 2176-5480

22nd International Congress of Mechanical Engineering (COBEM 2013)
November 3-7, 2013, Ribeirão Preto, SP, Brazil

A LATTICE BOLTZMANN METHOD SIMULATOR FOR MICROFLUIDICS ON GPU CLUSTER

Luiz Otávio S. Ferreira
Lucas Monteiro Volpe
UNIVERSITY OF CAMPINAS, Faculty of Mechanical Engineering, Department of Computational Mechanics, Rua Mendeleyev 200, CEP: 13083-860, Campinas, SP, Brazil
lotavio@fem.unicamp.br
lucasmvolpe@gmail.com

Abstract. *A simulator for microfluidic systems, based on lattice Boltzmann method (LBM) was developed for running on a Graphics Processing Unit (GPU) cluster. It was written on CUDA C language, implementing single component single phase fluids, and includes periodic, velocity, bounce-back and pressure boundary conditions. The program was run on a cluster with four node, where each node contains one quad-core CPU with 12 GB DDR3 2000 MHz memory, and four 512 cores NVIDIA GeForce GTX580, 1.5 GB GDDR5 GPUs. A simple on-line visualization program is used to follow-up the simulations, such that "on-the-flight" adjustments of the simulation parameters may be made. Our results show that interactive simulation on GPU accelerated the tuning of operational parameters of a microfluidic oscillator on the order of one tenth the time needed by an offline simulation on GPU. Given that a single GPU runs the simulator at least 30 faster than a CPU, the interactive simulator on a cluster of GPUs extends this advantage to problems on the order of tens of millions of lattice units.*

Keywords: *microfluidics, simulation, lattice-Boltzmann, particles, GPU*

1. INTRODUCTION

Fluid behavior in microfluidic devices has important phenomena that are neglected on macroscopic systems, such as surface tension and brownian motion (Zhang, 2010). And it is common that complex fluids be handled on microfluidic systems like Lab-On-a-Chip (LOC) or Micro-Total-Analysis-System (μ TAS), used for chemical and biochemical analysis. Such phenomena are difficult to incorporate on conventional computational fluid dynamics (CFD) simulation methods, but are succesfully treated by the lattice Boltzmann method (LBM). This method was proposed by McNamara and Zanetti in 1988, as a solution to the problems of the Lattice Gas Cellular Automata method (LGCM) (Wolf-Gladrow, 2005), and since then is receiving improvements such that today it is recognized as a reliable and efficient complex fluid simulation method (Aidun and Clausen, 2010), and is appropriate for microfluidic systems simulation (Zhang, 2010), with the extra advantage of being very efficient for parallel processing both on CPU clusters (Donath *et al.*, 2010) and on GPUs (Graphics Processing Units) (Obrecht and Kuznik, 2011; Obrecht *et al.*, 2011a,b, 2012; Astorino, M.; Becerra Sagredo, J.; Quarteroni, 2011). This work presents the implementation of a computational fluid dynamics simulator based on the lattice Boltzmann method, for interactive simulations of microfluidic devices, exploring the high performance of this method on GPUs.

2. MATERIALS AND METHODS

2.1 Simulation hardware

A cluster equipped with GPUs was used on this work. Each one of the four nodes of the cluster is equipped with four GPUs, and the internode communication is made using a gigabit ethernet switch.

2.1.1 Host CPU description

On each node there is a CPU model Intel Core i7 970 running at 3.2 GHz on an ASUS P7T6 WS Supercomputer mainboard. The local RAM is 12 GB DDR 3 @ 2000 MHz. There is no local hard disk. Two 1200 W power supplies are connected in parallel to the mainboard and its four associated GPUs, connected to four of its seven PCIe connectors at x16 datapath width. This nodes are diskless, remote boot.

2.1.2 GPU description

Four GeForce GTX580 GPU cards, each one equipped with 1.5 GB DDR5 RAM, are connected to each mainboard. These cards have 512 floating point and integer cores, allocated on 16 multiprocessors, and a peak processing power of 1.58 TFLOPs. These cards have CUDA computation capability 2.0, that includes unified virtual addressing (UVA),

ISSN 2176-5480

making the address space of all GPUs at the same mainboard look to have a continuous address space. It means that a thread running when the user press *u*on a GPU can access the memory of GPUs on the same mainboard like if it was its own memory, under the programmer viewpoint. It is very useful because the simulation domain is divided on subdomains, and each subdomain is processed on a different GPU, that must access the neighbors domains to calculate new values for the border points of its own subdomain.

2.2 System software

All GPU equipped nodes of the cluster run the desktop version of the Ubuntu 12.04 operating system. Another machine, equiped with hard disks, is the software serve and the administration node of the system. DRBL (Diskless Remote Boot in Linux), from the Free Software Lab[1] of the National Center for High-Performance Computing of Taiwan[2] is used on top of the operating system to manage the cluster. This server is based on NFS-/NIS filesystem, and offers diskless environments for client machines. It provides remote boot for the diskless GPU nodes and concentrates all the data storage of the cluster. By using DRBL it is possible to have a centralized management of the software for the entire cluster. All client machines were adjusted to startup from network using PXE protocol, and download an image of the system stored on DRBL server.

2.3 The lattice Boltzmann method

The application software simulates fluid dynamics using the attice Boltzmann method, that is a discrete version of the Boltzmann theory for transport processes in gases (Kremer, 2010). A fluid is modeled as fictitious particles moving between nodes of a regular lattice with discrete velocities directions at sequential time steps, where on each time step the collision and streaming of particles are calculated sequentially using Eq.(1)

$$f_a(\mathbf{x} + \mathbf{e}_a \Delta t, t + \Delta t) = f_a(\mathbf{x}, t) - \frac{[f_a(\mathbf{x}, t) - f_a^{eq}(\mathbf{x}, t)]}{\tau} \tag{1}$$

Where f is a distribution function, $\mathbf{x}$ is the position of particle, $\mathbf{e}_a$ is its microscopic velocity, t is time, f_a^{eq} is the equilibrium distribution, $\mathbf{\Delta}t$ is the time-step of simulation, τ is the relaxation time, $f_a(\mathbf{x} + \mathbf{e}_a \Delta t, t + \Delta t) = f_a(\mathbf{x}, t)$ is the streaming part and $[f_a(\mathbf{x}, t) - f_a^{eq}(\mathbf{x}, t)]/\tau$ is the collision term.

The collision term of Eq.1 is a simplified solution (BGK) of the Boltzmann equation, introduced in 1954 by Bhatnagar, Gross and Krook (Mohamad, 2011). Figure 1 shows the $D2Q9$ lattice used on this work (Sukop and Thorne Jr., 2005), where $D2$ means it is a *2D* lattice, and $Q9$ means it has 9 discrete velocity vectors, with central vector of speed zero.

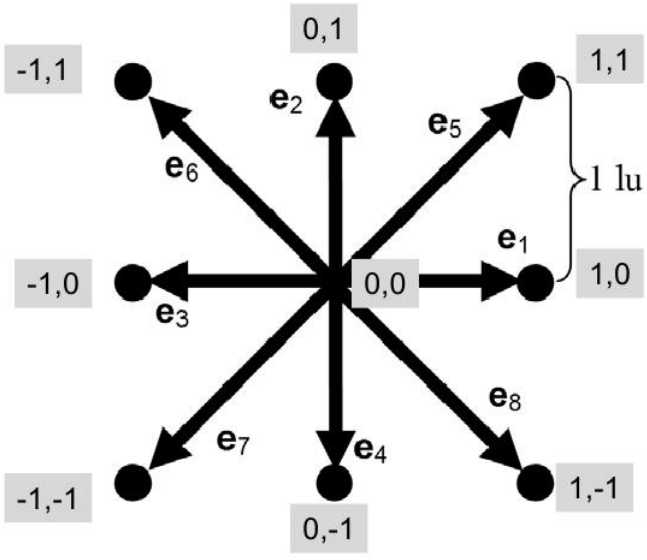

Figure 1. D2Q9 lattice cell, with 9 velocities connecting each note to its neighbors (Sukop and Thorne Jr., 2005). The microscopic velocities e_0 to e_8 and their corresponding (x, y) components are shown.

[1] http://free.nchc.org.tw/pmwiki/index.php?n=Main.HomePage
[2] http://www.nchc.org.tw/en

ISSN 2176-5480

And the equilibrium distribution of the BGK scheme is given by Eq. (2)

$$f_a^{eq}(\mathbf{x}) = w_a \rho \left[1 + \rho_0 \left(3\frac{\mathbf{e_a}\cdot \mathbf{u}}{c^2} + \frac{9}{2}\frac{(\mathbf{e_a}\cdot \mathbf{u})^2}{c^4} - \frac{3}{2}\frac{\mathbf{u^2}}{c^2} \right) \right] \tag{2}$$

Where $\rho_0 = 1$ and the weighting factors w_a are: $4/9$ for $\mathbf{e_0}$; $1/9$ for $\mathbf{e_1}$ to $\mathbf{e_4}$; and $1/36$ for $\mathbf{e_5}$ to $\mathbf{e_8}$.
Macroscopic density of fluid is given by Eq. (3)

$$\rho = \sum_{a=0}^{8} f_a \tag{3}$$

Equation 4 is its macroscopic velocity $\mathbf{u}$, an average of the microscopic velocities weighted by the probability density f_a

$$\mathbf{u} = \frac{1}{\rho}\sum_{a=0}^{+8} f_a \mathbf{e_a} \tag{4}$$

2.4 Simulation software

C++ and CUDA C, that is an extension of the C language for NVIDIA ® GPUs, were used to code the simulation software. All CUDA C functions were embedded on C++ classes. Four different types of boundary conditions were used on this work: periodic, bounceback, constant pressure and constant velocity, as detailed on Sukop and Thorne Jr. (2005).

2.4.1 Main loop

Simulation begins by reading the following input parameters from a text file: width of domain (x); height of domain (y); x axis velocity; y axis velocity; viscosity; total time steps; and time steps between saving simulation status. Then a file that contains a bitmap that describes the simulation domain is read. Solid nodes are black pixels and fluid nodes are white pixels. Then is made the setup of the initial conditions, and the program enters a loop where it calculates the macroscopic density and velocities, equilibrium distribution, apply the boundary conditions, performs collision and streaming calculation.

2.4.2 InterGPU communication

Simulation domain is divided on subdomains, and each subdomain is processed on a different GPU, that must access the neighbors domains to calculate new values for the border points. If the GPUs are on the same node, they exchange domain border data by DMA, without interference of CPU, using the regular CUDA *cudaMemcpy()* function, once passed the parameter *cudaMemcpyDefault*, given that the hardware has compute capability 2.0 or upper. Under the programmer viewpoint, the address space of the same node GPUs are part of a common range of addresses. If the neighbors subdomains are on different node of the cluster, the UVA is not valid. The solution is to use the MPI library, as described on the next topic.

2.4.3 Internode communication

When subdomain data are transfered between GPUs of different nodes, first the border data is transfered to the local CPU, then it is transfered to the CPU of the destination node by MPI functions *MPI_Sendrecv_replace()*, and then the CPU of the destination node transfers the data to local GPUs using the CUDA function *cudaMemcpy()*. Figure 2 shows the schemeThe list of references must be introduced as a new section, located at the end of the paper. The first line of each reference must be aligned at left. All the other lines must be indented by 0.5 cm from the left margin. All references included in the reference list must have been mentioned in the text.

References must be listed in alphabetical order, according to the last name of the first author. See the following examples: of message passing for domain border data exchange between GPUs during the streaming step of the simulation, where *fa*(on top) is the distribution function array before steaming step, and *fa_new*(on bottom) is the desired distribution function array configuration after the streaming step. The *streaming()* function applies the streaming step on each subdomain, that is processed as periodic boundaries. The *get_border()* function gets the borders data from *fa_new* to array *border_fa*. Function *copy_border_devices()* copies *border_fa* to array *nb_border_fa*, exchanging borders data between GPUs on the same node. Function *copy_border_nodes()* exchange the data between the MPI processes according to the mapping of subdomains onto GPUs. Finally function *apply_border()* transfers borders data to *fa_new* array, finishing the streaming step.

ISSN 2176-5480

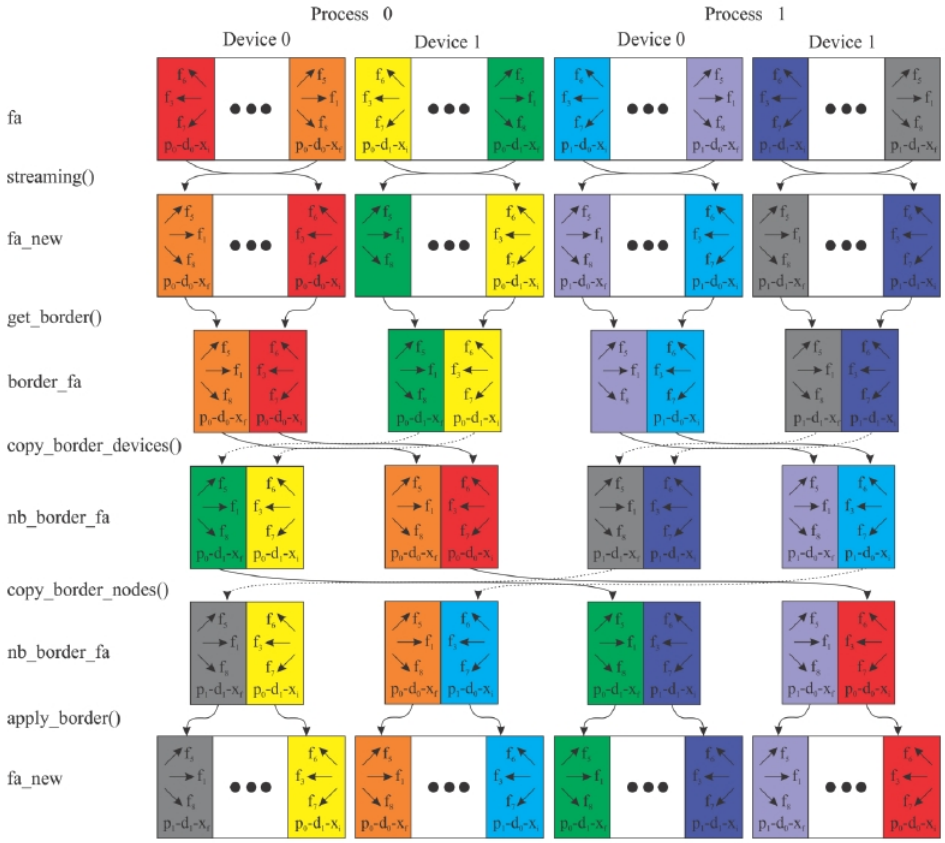

Figure 2. Data exchange between neighbors during streaming step for a cluster with 2 MPI process with 2 GPUs in each process. See details on text.

2.4.4 User interface

A graphical user interface based on OpenGL allows user to visualize colormaps of fluid velocity and pressure and to change parameters like viscosity, inlet velocity and outlet pressure during the simulation. These parameters are changed multiplying the actual value by a factor when the user press one of the following keys: u for inlet velocity, p for outlet pressure and v for viscosity. The multiplication factor is changed by pressing + or - keys.

3. RESULTS AND DISCUSSION

At first the fluid dynamic behavior of the simulator was validate by running classical Hagen-Poisseuille flow shown on Fig. 3 on a single GPU workstation and checking the parabolic velocity profile of the flow.

Then a microfluidic oscillator (Fig.4) was simulated on the same workstation to test the functionality of the interactive simulation on faster verification of a new design. The target was to find the oscillation condition of the new device. It was verified that through the interactive simulation it was possible to find the right oscillation condition without restarting the simulation each time a parameter was changed. This design tuning process demonstrated to be as least one order of magnitude faster than the traditional batch simulation. The interactive character of the simulator was possible thanks to GPU acceleration, because while the GPU performance was of 473 MLUPS for the Hagen-Poisseille flow, CPU performance was only 22 MLUPS.

Then the simulator was run on the cluster and its performance for several problems sizes is shown on Fig. 5, where we can see that for problems size larger then 70,000 lu^2 there is a performance increase adding more GPUs on a single node. There is advantage on running the simulator on more then one node when the problem size is larger than 600,000 lu^2. This results can be explained by communication cost. In the same node the communication cost is caused by PCIe limited bandwidth of 16 GB/s, while internode communication have only 1 GB/s bandwidth and much higher latency. As each GPU has 1.5 GB of memory, larger problems may be run only by domain partitioning between several GPUs. Problems up to 314 million lu^2 was run succesfully on our cluster.

ISSN 2176-5480

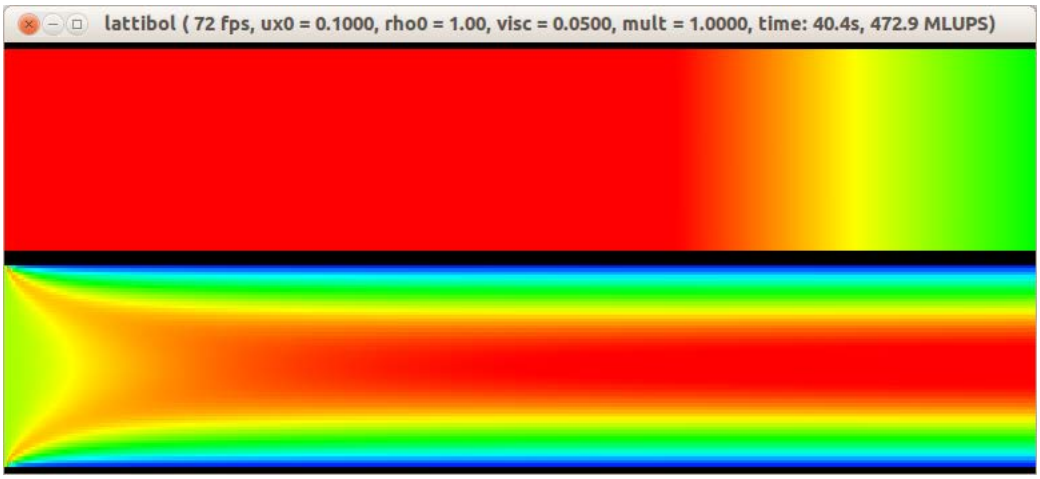

Figure 3. Print screen of a Hagen-Poisseuille flow simulation. On the upper half is the pressure colormap and on the bottom half is the velocity colormap. On window top are, from left to right: program name (lattibol), frame rate, inlet velocity, outlet density, viscosity, multiplication factor, execution time and performance figure in MLUPS (millions lattice updates per second).

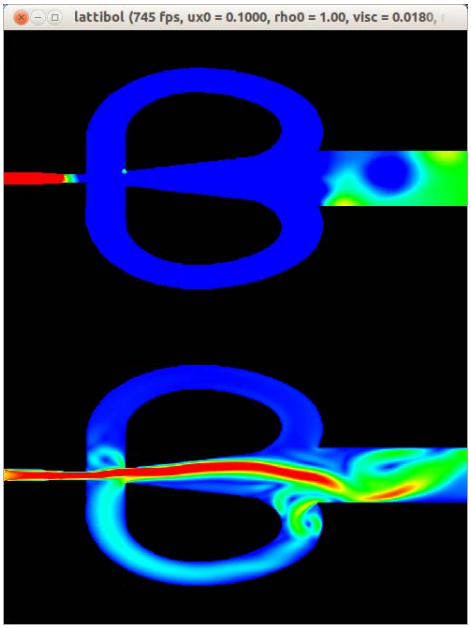

Figure 4. Microfluidic oscillator simulation (drawing adapted from http://aes.cm.kyushu-u.ac.jp/mixing/fluidic07n.htm)

4. CONCLUSION

A fluid dynamics simulator based on lattice Boltzmann method is been developed for the special purpose of interactive simulation of microfluidics systems. The interactive character of this simulator is important because on the development cycle of a microfluidic device there is need for adjustment of several operational parameters based on its performance under given conditions. By interactive visualization and change of the parameters, it was possible to reach the target behavior at least on one tenth of the time if an offline simulator were used. Larger problems could be speedup by distributed processing on a cluster of GPUs, where the speedup relative to one GPU was 3.8 using 4 GPUs on a single node and was 7.7 using 16 GPUs on 4 nodes. The speedup limiting factor is the communication bandwidth between nodes.

Next steps on the development of the simulator are the replacement of actual user interface for a QT based one and the addition of virtual instruments for punctual measurement of parameters like velocity and pressure.

5. ACKNOWLEDGEMENTS

This work was supported by CAPES, by CNPQ under process number 310474/2009-4, and by FAPESP under process number 2010/09717-7.

ISSN 2176-5480

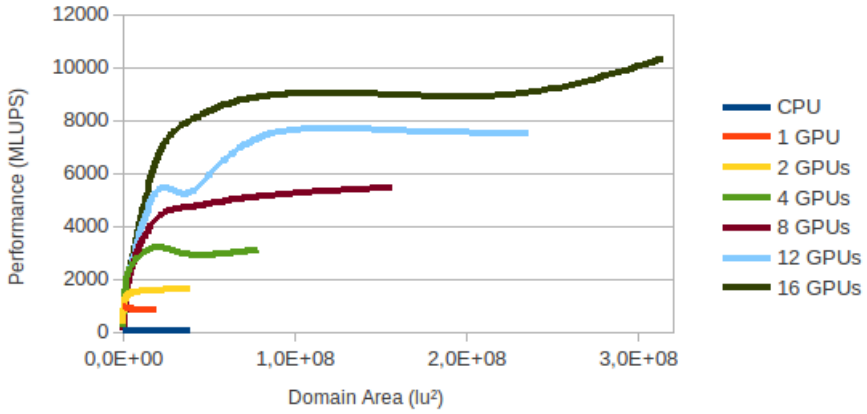

Figure 5. Performance on GPU cluster.

6. REFERENCES

Aidun, C.K. and Clausen, J.R., 2010. "Lattice-Boltzmann Method for Complex Flows". *Annual Review of Fluid Mechanics*, Vol. 42, No. 1, pp. 439–472.

Astorino, M.; Becerra Sagredo, J.; Quarteroni, A., 2011. "A modular lattice Boltzmann solver for GPU computing processors". *MOX-Report*, , No. 31.

Donath, S., Feichtinger, C. and Pohl, T., 2010. "A Parallel Free Surface Lattice Boltzmann Method for Large-Scale Applications". *Parallel Computational . . .* , pp. 3–7.

Kremer, G., 2010. *An introduction to the Boltzmann equation and transport processes in gases*. Springer.

Mohamad, A.A., 2011. *Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes*. Springer.

Obrecht, C. and Kuznik, F., 2011. "Global memory access modelling for efficient implementation of the lattice Boltzmann method on graphics processing units". In *Proceedings of the 9th international conference on High performance computing for computational science*. pp. 151–161.

Obrecht, C., Kuznik, F., Tourancheau, B. and Roux, J.J., 2011a. "The TheLMA project: Multi-GPU implementation of the lattice Boltzmann method". *International Journal of High Performance Computing Applications*, Vol. 25, No. 3, pp. 295–303.

Obrecht, C., Kuznik, F., Tourancheau, B. and Roux, J.J., 2011b. "A new approach to the lattice Boltzmann method for graphics processing units". *Computers & Mathematics with Applications*, Vol. 61, No. 12, pp. 3628–3638.

Obrecht, C., Kuznik, F., Tourancheau, B. and Roux, J.J., 2012. "The TheLMA project: A thermal lattice Boltzmann solver for the GPU". *Computers & Fluids*, Vol. 54, pp. 118–126.

Sukop, M.C. and Thorne Jr., D.T., 2005. *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers*. Springer.

Wolf-Gladrow, D.A., 2005. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models - An Introduction*. Springer.

Zhang, J., 2010. "Lattice Boltzmann method for microfluidics: models and applications". *Microfluidics and Nanofluidics*, Vol. 10, No. 1, pp. 1–28. ISSN 1613-4982.

7. RESPONSIBILITY NOTICE

The following text, properly adapted to the number of authors, must be included in the last section of the paper: The authors are the only responsible for the printed material included in this paper.