8th ICCHMT, Istanbul, 25–28 May 2015

138

ANALYSIS OF HEAT TRANSFER IN SLIP-FLOW REGIME BY PARALLEL IMPLEMENTATION OF LATTICE-BOLTZMANN METHOD on GPUs

Barbaros Çetin[1]*, S. Berat Çelik[2], Cüneyt Sert[2]
[1]Microfluidics & Lab-on-a-chip Research Group, Mech. Eng. Dept.
İhsan Doğramacı Bilkent University, 06800 Ankara, Turkey
[2]Dept. Mechanical Eng., Middle East Technical University, Ankara 06800 TURKEY
*Correspondence author: Phone: +90-312-290-2108, Email: barbaros.cetin@bilkent.edu.tr
Keywords: Lattice-Boltzmann Method, slip-flow, GPU Computing

ABSTRACT

In this study, fluid flow and heat transfer in two-dimensional microchannels are studied numerically. A computer code based on Lattice Boltzmann Method (LBM) is developed for this purpose. The code is written by using MATLAB and `Jacket` softwares and has the important feature of being able to run parallel on Graphics Processing Units (GPUs). Obtained velocity profiles and Nusselt numbers are compared with the Navier-Stokes based analytical and numerical results available in the literature and good matches are observed. Slip-velocity and temperature-jump boundary conditions are used for the microchannel simulations with Knudsen number values covering the slip-flow regime. Speed of the parallel version of the developed code running on GPUs is compared with that of the serial one running on CPU and for large enough meshes more than 14 times speed-up is observed.

NOMENCLATURE

f	distribution function
f^{eq}	Equilibrium distribution function
g	thermal distribution function
g^{eq}	thermal equilibrium distribution function
H	channel height
k	thermal conductivity
Kn	Knudsen number
Nu	Nusselt number
Pr	Prandtl number
$\mathbf{v}$	velocity field
v^{slip}	slip velocity
v_{mean}	mean velocity
w_i	weight functions
α	thermal accommodation factor
α_{th}	lattice thermal diffusivity
κ	parameter defined in Eq. (13)
ν	lattice kinematic viscosity
γ	specific heat ratio
ω	collision frequency
ω_{th}	collision frequency for thermal LBM

ϕ	lattice temperature
σ	momentum-accommodation factor
ρ	density

INTRODUCTION

The physics of fluid flows can be modeled by several different set of mathematical equations. All those different sets are somehow successful even the logic behind them are completely different. The logical approaches to physical phenomena can split in two major topics: Continuum models and particle models. Continuum assumption states that every single point in a flow field has a finite physical property (temperature, pressure, density etc.) by accepting fluids to be divisible into sub-fluids indefinitely. Hence, there occurs no discontinuity and the assumption is called continuum. The flow field needs to be in thermodynamic equilibrium. By thermodynamic equilibrium, it is understood that there is always enough time for particles (molecules/atoms) to adjust themselves according to surrounding variations. Thermodynamic equilibrium assumption spawns no-slip and no-temperature-jump boundary conditions. The continuum and equilibrium assumptions may break down in some conditions. In MEMS, it is common not to have continuum and thermodynamic equilibrium due to the tiny length scales of the fluid channels. If we describe λ to be the average distance of molecules migrate between two successive collisions, Knudsen number (Kn) is the ratio λ/H, where H is the characteristic length of the system, usually the hydraulic diameter of the flow channel. Increasing Kn is either due to increase in molecular distance or due to decrease in characteristic length. Increasing Kn violates the assumptions of the continuum model equations. Therefore different approaches are needed to model fluid flow at high Knudsen numbers [1].

The Lattice Boltzmann method (LBM) is also a statistical simulation tool. LBM originates from Cellular Gas Automata which was not a successful tool until the idea of using it to solve Boltzmann Transport Equation (BTE). In theory, BTE is capable to cover all flows with Kn ranging from zero to infinity. In this study, fluid flow and heat transfer in a microchannel in slip-flow regime is simulated using LBM with D2Q9 model and BGK collision approximation. Momentum equation and energy equation together with constant wall temperature boundary condition for Poiseuille flow in a microchannel are solved. Computations are performed by using MATLAB on a desktop computer with a Tesla C1060 GPU. To utilize the parallelization potential of LBM, performance comparison with CPU-based and GPU-based computing is presented. For GPU-computing both single-GPU and multi-GPU computing is tested.

THEORETICAL ANALYSIS

Lattice Boltzmann Method (LBM): The governing equation of LBM is BTE. BTE can be written in discrete form by introducing Bhatnagar-Gross-Krook (BGK) model [2], and D2Q9 model as:

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \frac{\partial f_i}{\partial \mathbf{x}} = \omega(f_i^{eq} - f_i), \tag{1}$$

where $i = 1, 2, ..., 9$ and $\mathbf{e}_i$ refer to the discrete velocities for D2Q9 model [3], and f^{eq} is a Maxwellian equilibrium distribution function [4]. On a rectangular mesh ($\Delta x = \Delta y$), the spatial derivative of Eq. (1) can be approximated by a first order upwind difference and the time derivative with a first order explicit difference with time step Δt ($= \Delta x = \Delta y = 1$), the equation reduces to

$$f_i(\mathbf{x}+\Delta t\mathbf{e}_i, t+\Delta t) = \omega f_i^{eq} + (1-\omega) f_i(\mathbf{x}, t). \quad (2)$$

The equilibrium function, f^{eq}, ω and the macroscopic variables can be written as [2]:

$$f_i^{eq} = w_i \rho [1 + 3(\mathbf{e}_i \cdot \mathbf{v}) + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{v})^2 - \frac{3}{2}|\mathbf{v}|^2. \quad (3)$$

The collision frequency is defined as [3]:

$$\omega = \frac{1}{3\nu + 0.5} \quad (4)$$

w_i's are the weight functions and can be written as:

$$w_k = \begin{cases} 1/9 & \text{for } k = 1,2,3,4 \\ 1/36 & \text{for } k = 5,6,7,8 \\ 4/9 & \text{for } k = 9 \end{cases} \quad (5)$$

Eqs. (2)–(5) are the governing equations for LBM. ω is defined only as a function of lattice kinematic viscosity for incompressible flows [2].

Thermal LBM: Besides the flow of a fluid, LBM is also capable to solve temperature distribution in fluids. The procedure is to solve the fluid flow and the temperature respectively in the same time step. The advection-diffusion equation is obtained by the thermal distribution function, g [2]:

$$\frac{\partial g_i}{\partial t} + \mathbf{e}_i \cdot \frac{\partial g_i}{\partial \mathbf{x}} = \omega_{th}(g_i^{eq} - g_i) \quad (6)$$

The equilibrium function, g^{eq}, ω_{th} and the macroscopic variable can be written as [2]:

$$g_i^{eq} = w_i \phi(\mathbf{x}, t)\,[1 + 3(\mathbf{e}_i \cdot \mathbf{v})] \quad (7)$$

The thermal collision frequency is given as [3]

$$\omega_{th} = \frac{1}{3\alpha_{th} + 0.5} \quad (8)$$

The weight functions, w_i's, are the same as momentum weight functions, ω_{th} is collision frequency for thermal LBM, α_{th} is lattice thermal diffusivity and $\phi(\mathbf{x}, t)$ is lattice temperature.

Collision & Streaming: LBM can be considered to be composed of 4 different parts, respectively: Collision, streaming, implementation of boundary conditions and calculation of macroscopic properties. Streaming (Updating) process is, actually, a part of collision calculations. However, it is simpler if handled separately. During the collision calculations, the function can be calculated as follows:

$$f_i(\mathbf{x}, t+\Delta t) = \omega f_i^{eq} + (1-\omega) f_i(\mathbf{x}, t) \quad (9)$$

which results in the distribution functions of the next time step. Remember that there is also a spatial discretization. This was performed by inserting the value of distribution functions from a neighboring node

$$f_i(\mathbf{x}+\Delta\mathbf{x}, t+\Delta t) = f_i(\mathbf{x}, t+\Delta t), \quad (10)$$

and this process is addressed as streaming in LBM. Details about the streaming process can be found elsewhere [3, pp. 23–24].

Boundary Conditions: At the inlet, uniform velocity was assigned. Contrary to the inlet boundary condition, the velocity at the outlet was unknown. However, we could assume that the velocity is no more varying on the stream-wise direction in a long enough channel. That is equivalent to say the flow is fully developed. Assuming the velocity distribution close to the exit do not change, we could safely assign the velocity of the nodes close to exit to that of the exit nodes (details of the derivation of the boundary conditions can be found elsewhere [3, pp. 24–29]). Thermal boundary conditions were determined using the equality of the non-equilibrium distribution functions [2]. At the inlet, uniform temperature was assigned. The outlet temperature was calculated by extrapolating the thermal distribution functions. The boundary conditions at the channel wall requires special care due to the small Kn characteristics of the flow in a microchannel. To take into account the rarefaction effect, slip-velocity and temperature-jump boundary conditions needed to be used instead of no-slip and no-temperature-jump boundary conditions of flow in macrochannels.

Boundary Conditions for Slip-flow: First order velocity slip for a stationary wall can be written as [5]:

$$v_{y=0}^{slip} = \sigma Kn \left(\frac{\partial v}{\partial y}\right)_{y=0} \tag{11a}$$

$$v_{y=H}^{slip} = \sigma Kn \left(\frac{\partial v}{\partial y}\right)_{y=H} \tag{11b}$$

where σ is assumed to be unity which is the case for many gas-solid pairs used in engineering applications [6]. Similarly, the temperature-jump boundary conditions can be written as [5]:

$$\phi_{y=0} - \phi_w = \kappa Kn \left(\frac{\partial \phi}{\partial y}\right)_{y=0} \tag{12a}$$

$$\phi_w - \phi_{y=H} = \kappa Kn \left(\frac{\partial \phi}{\partial y}\right)_{y=H} \tag{12b}$$

where κ is defined as:

$$\kappa = \alpha \frac{2\gamma}{\gamma+1}\frac{1}{Pr} \tag{13}$$

where α is taken as one.

Calculation of Macroscopic Properties: Density and velocity of the flow field were calculated using the mass and momentum conservations for each node.

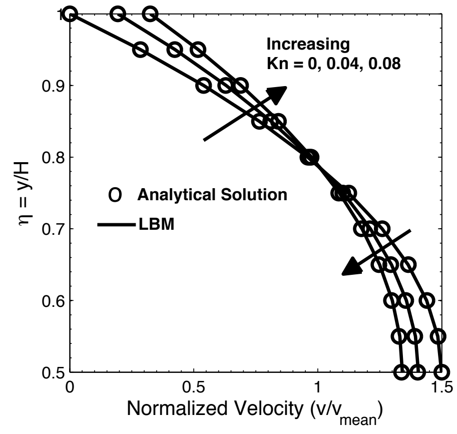


Figure 1
Fully-developed velocity profile

$$\rho(\mathbf{x},t) = \sum_{i=1}^{9} f_i(\mathbf{x},t) \tag{14}$$

$$\mathbf{v}(\mathbf{x},t) = \frac{\sum_{i=1}^{9} \mathbf{e}_i f_i(\mathbf{x},t)}{\rho(\mathbf{x},t)} \tag{15}$$

The temperature, on the other hand, was calculated using the thermal distribution function.

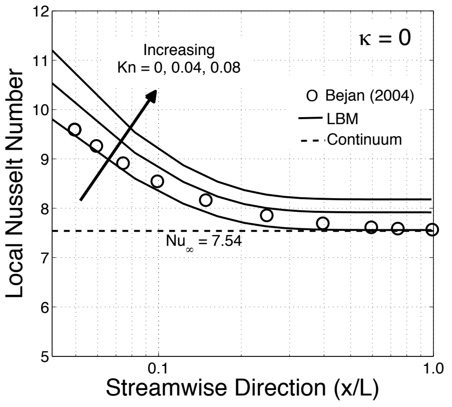
$$\phi(\mathbf{x},t) = \sum_{i=1}^{9} g_i(\mathbf{x},t) \tag{16}$$

Knowing the temperature field, local Nu can be determined using the approprite formulations [3].

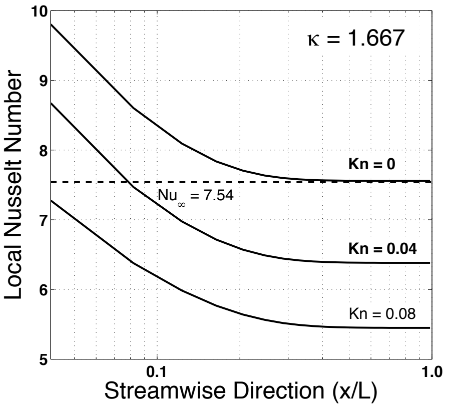
LBM RESULTS

An LBM code was developed to simulate some benchmark problems at macroscale [7,8]. Then, the code was modified to simulate microflows in the slip-flow regime. Kn was varied between 0 and 0.08 which are the within the applicability limits of the slip-flow regime. Parameter κ was taken as 0, and 1.667. A grid convergence test was also performed. A set of runs for several different mesh resolution indicated that the solution is converged to the expected values. As a result of the convergence test, 61×1620 grid was used in the simulations. Aspect ratio was taken as 20 to ensure the fully-developed velocity and the fully-developed temperature. The velocity profile for different Kn were plotted together with the analytical solution available in the literature [6] in Fig. 1. Perfect agreement was observed.

The Nu depends on the velocity and temperature profiles. For the developing part of the flow, Nu curves seem to converge to a value as soon as both the velocity and temperature become fully-developed. Results were obtained for different κ and Kn. Fig. 2(a) shows local Nu variation along the channel for $\kappa = 0$ and for $Kn = 0$, 0.04, 0.08. Well-known macro-channel results were also included from [9] in the figure. $Kn = 0$ case converges to the well-known analytical result for macrochannels ($Nu = 7.54$). Increasing Kn results in higher Nu. Note that $\kappa = 0$ indicates that there exists no temperature jump which is a fictitious case but is used in data validation in the literature [6]. Fig. 2(b) has the results for $\kappa = 1.667$ which is a typical value for air which is the working fluid in many engineering applications.



(a) $\kappa = 0$



(b) $\kappa = 1.667$

Figure 2
Local Nu along the microhannel

COMPUTING

Central Processing Units (CPU) are where the mathematical operations are performed and they are designed to perform operations in a sequential order. The use of GPUs for general purpose parallel scientific computing is seen to be a promising way of accelerating number crunching codes. Unlike CPUs, GPUs can accommodate hundreds of processors without any extreme heat generation. A drawback of GPU computing was the complexity of the programming languages. There had been several languages and most of them were abandoned. After the invention of CUDA and OpenCL programming tools, researchers no more need to

Table 1
Comparison of fully-developed Nu

Knudsen	$\kappa = 0$		$\kappa = 1.667$	
(Kn)	Nu_∞	Nu^*_∞	Nu_∞	Nu^*_∞
0	7.55	7.54	7.55	7.54
0.04	7.91	7.91	6.38	6.37
0.08	8.18	8.18	5.44	5.45

Nu_∞: Present study
Nu^*_∞: Results from [6]

be experts in computer graphics to harness the computational power of GPUs [10]. However, the nature of parallel computing is not similar to serial programming and needs **(i)** a different approach to the design of algorithms, **(ii)** a familiarity to cache memory, and **(iii)** a proper distribution of the calculations to the cores. Many researchers, during the last decade, tested LBM on parallel computing and reported their findings. Tolke developed an LBM code using CUDA. He wrote his code in C language to run on a single GPU and obtained 1 order of magnitude speed-up compared to his serially running LBM code [11]. Obrecht et al. [12] wrote a 3DQ19 LBM code and ran it on a nVidia GTX295 GPU. They obtained nearly 2 fold increase. Riegel et al. [13] developed an LBM solver called LBultra, and tested it for the 3D benchmark problem of flow over a cylinder. Their code written in C++ ran on 3 Tesla C1060 GPUs, and they reported about 19 fold increase in speed. Their parallel CPU code ran on 4 AMD cores or 2 Intel cores but those could not approach the speed of multi-GPU code. 4 AMD cores provided 1.8 fold speed-up, and 2 Intel cores provided 2 fold of increase in speed. They also observed that GPU programming saves more energy, space and money than parallel CPU computing.

In this work, the coding was performed in MATLAB environment. MATLAB is serial inherently; and hence, works on CPU. Another program named as `Jacket`, which was created by AccelerEyes (Atlanta, GA, USA) company, is an add-on application to MATLAB and allows MATLAB to perform mathematical operations on GPUs. Basically, `Jacket` is a link from MATLAB to GPU programming through the use of CUDA technology. The advantage of Jacket is the ease of use. With a very little afford, the standard MATLAB gets ready to run on GPU. For the Poiseuille flow benchmark problem, 3 different programs were developed. First code solved the problem serially on a single core of a CPU. The CPU used is Intel Xeon E5620 Quad-Core 2.40 GHz. Second code run parallel on a single GPU that is Tesla C1060. Tesla C1060 has 240 cores and capable of providing 933 GFLOPs/s of performance with 4 GB of GDDR3 memory at 102 GB/s bandwidth. The last one was parallel too, yet distributed among 4 Tesla C1060s and referred as Multi GPU (MGPU) version. The parameters used in the simulations were: $Re = 10$, $Pr = 10$, $Kn = 0$, aspect ratio = 20. The solution domain was discretized into $M \times N$ grid points. 9 distribution functions which were $M \times N$ in size were allocated for each collision part. Even though the operation on one distribution function does not affect the other distribution functions on the same or other grid points, MATLAB performs these calculations serially. However, `Jacket` distributes the operations of one distribution function among 240 cores of the GPU. In other words, the domain was divided into 240 sub-domains for each distribution function. The Multi GPU code had a

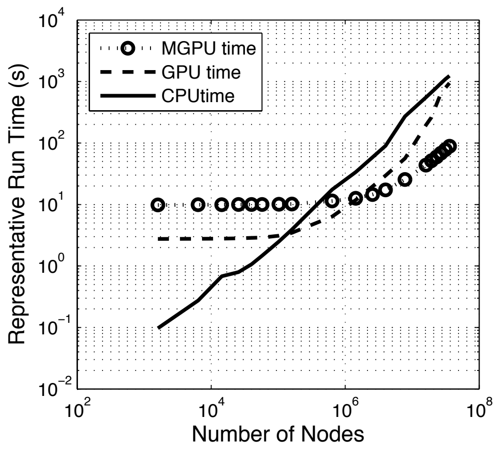


Figure 3
MGPU, GPU and CPU run times

little different structure than the CPU and single GPU code. In order to utilize all 4 GPUs, the domain was divided into 4 parts. Every part was solved on a different GPU. Due to the fact that every device has its own memory, the variables should be allocated on each device with different names. After the collision, the nodes on GPU boundaries were updated using ghost nodes. The update was a data exchange between neighboring GPUs. The data was first transferred to CPU memory and than to the other GPU.

The computational time demonstrated in Fig. 3 is not the convergence time but rather it is a time that covers a smaller number of time steps which is enough to compare the speeds. It is seen that with the increasing mesh density CPU time increase continuously. Single GPU parallel code has a constant elapsed time up to a mesh size of 10^5. Before that mesh size, single GPU consumes more time than a single CPU core. After a mesh size of 3x10^7 the GPU time approaches to CPU time due to memory issues. The memory of CPU is 24GB whereas a single Tesla C1060 has only 4GB memory. In other words, for mesh sizes greater than 10^5 and smaller than 3x10^7, it is feasible to run on GPU. The curve for MGPU is steady up to a mesh size of 10^6. After this point both CPU and GPU times increase faster, so MGPU can be used for meshes greater than 10^6.

The GPU and MGPU programs have constant speeds for smaller mesh sizes. This is a common picture in GPU computing. Even though the code run in GPU, there should be variable transfer between CPU memory and GPU memory. The time due to variable transfer is dominant for small mesh sizes. In the Fig. 3, most of the time spent in the steady region was due to the variable transfer. A mesh size of 10^5 for GPU and 10^6 for MGPU programs were the limits where the time spent on computation begins to overcome the time spent on variable transfer. Due to this fact, domains with small mesh number cannot be accelerated; also, they run slower than serial computing. Compared to the serial program, the single GPU parallel program might consume about 5 fold less time at 7.84x10^6 grid points and 14 fold increase in the speed of MGPU program was observed for the maximum mesh size. The memory constraints the computational speed for larger mesh sizes. In a communication process, the data is transferred to GPU from CPU and vice versa. When the variable transfer is performed to 4 GPUs, 4 times more data can be transferred at a time compared to a transfer to a single GPU. That means a single GPU has to perform 3 more transfer processes and these transfer processes consumes considerable time. Hence, there could occur speed-ups more than 4 for very large mesh sizes. These communication processes are the major drawbacks of parallel programming on GPUs.

ACKNOWLEDGEMENT

Financial support from the Turkish Scientific and Technical Research Council, Grant No. 110M750, is greatly appreciated.

REFERENCES

1. Gad-elHak, M., 2008. *Advances in Multiphysics Simulation and Experimental Testing of MEMS.* Imperial College Press.

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

3. Celik, S. B., 2012. "Analysis of single-phase fluid flow and heat transfer in slip-flow regime by parallel implementation of Lattice-Boltzmann method on GPUs". Master's thesis, Middle East Technical University, Ankara, Turkey.

4. Qian, Y. H., D'Humieres, D., and Lallemand, P., 1992. "Lattice BGK Models for Navier-Stokes Equations". *Europhysics Letters,* **17**(6), pp. 479–484.

5. Tian, Z., Zou, C., Liu, Z., Guo, Z., Liu, H., and Zheng, C., 2006. "Lattice boltzmann method in simulation of thermal micro-flow with temperature jump". *Int. J. Modern Physc. C,* **17**(5), pp. 603–614.

6. Cetin, B., Yuncu, H., and Kakac, S., 2006. "Gaseous flow in microconduits with viscous dissipation". *Int. J. Transport Phenomena,* **8**, pp. 297–315.

7. Celik, S. B., Sert, C., and Cetin, B., 2011. "Simulation of lid-driven cavity flow by parallel implementation of Lattice–Boltzmann method on GPUs". In 2nd International Symposium on Computing in Science and Engineering, June 1–4, Turkey.

8. Celik, S. B., Sert, C., and Cetin, B., 2011. "Simulation of channel flow by parallel implementation of thermal Lattice-Boltzmann method on GPUs". In ASME 7th International Conference on Computational Heat and Mass Transfer, July 18–22, Istanbul, Turkey.

9. Bejan, A., 2004. *Convective Heat Transfer.* John Wiley & Sons.

10. Stone, J. E., Hardy, D. J., Ufimtsev, I. S., and Schulten, K., 2010. "GPU-accelerated molecular modelling coming of age". *Journal of Molecular Graphics and Modelling,* **29**, pp. 116–125.

11. Tolke, J., 2010. "Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVidia". *Comput Visual Sci,* **13**, pp. 29–39.

12. Obrecht, C., Kuznik, F., Tourancheau, B., and Roux, J. J., 2011. "A new approach to the Lattice Boltzmann Method for Graphics Processing Units". *Computers and Mathematics with Applications,* **61**, pp. 3628–3638.

13. Riegel, E., Indinger, T., and Adams, N. A., 2009. "Implementation of a Lattice Boltzmann Method for numerical fluid mechanics using the nVIDIA CUDA technology". *CSRD,* **23**, pp. 241–247.