Comput. Methods Appl. Mech. Engrg. 225–228 (2012) 65–73

Contents lists available at SciVerse ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

GPU implementation of lattice Boltzmann method for flows with curved boundaries

Hao Zhou *, Guiyuan Mo, Feng Wu, Jiapei Zhao, Miao Rui, Kefa Cen

State Key Laboratory of Clean Energy Utilization, Institute for Thermal Power Engineering, Zhejiang University, Hangzhou 310027, PR China

ARTICLE INFO

Article history:
Received 25 November 2010
Received in revised form 25 October 2011
Accepted 12 March 2012
Available online 20 March 2012

Keywords:
Lattice Boltzmann method
GPU
CUDA
Curved boundaries

ABSTRACT

An efficient implementation of the lattice Boltzmann method (LBM) using the compute unified device architecture (CUDA) provided by nVidia was presented to simulate flows with curved boundaries. The flow around a circular cylinder was investigated as a typical case and satisfactory results in terms of precision and performance were obtained. The predicted drag coefficient (C_D), lift coefficient (C_l) and Strouhal number (S_t) agreed well with the results of previous related studies. The test results indicate that the number of threads has a great influence on the performance of graphical processing unit (GPU) based implementation. A total of 128 threads per block can lead to the best performance with nearly an 18-fold speed increase.

1. Introduction

The lattice Boltzmann method (LBM) originated from lattice gas automata (LGA) [1,2], and can also be derived directly from the Boltzmann equation formulated by Ludwig Boltzmann [3]. A direct connection between the Boltzmann equation and the incompressible Navier–Stokes equation can be established under the nearly incompressible condition [4,5]. LBM has been successfully used in a variety of configurations, even in those with complex coupling of physical and chemical processes [6]. LBM is a relatively new computational fluid method. Compared with the traditional computational fluid dynamics (CFD) such as the Navier–Stokes equation, LBM has a higher calculation efficiency, and is easy to realize parallel implementation and able to handle moving and complex boundaries [7].

Mesh refinement is always an effective means to improve simulation results. However, it usually results in a dramatic increase in computational complexity. Graphical processing unit (GPU) is specifically designed to be extremely rapid at processing large graphics data sets for rendering tasks. As the computational power of GPUs has exceeded that of PC-based central processing units (CPUs) by more than one order of magnitude while being available for a comparable price, the employment of GPUs to accelerate non-graphics computations has drawn much attention [8–13]. Both a single GPU [10] and a GPU cluster [14] have been used to accelerate computations in the field of computational physics. A significant speedup of GPU-based computation over traditional CPU-based computation has been reported in different areas. Li et al. [15] accelerated the computation of the LBM on general-purpose graphics hardware. Tolke [16] reported an efficient implementation of a 2D-lattice Boltzmann kernel using the compute unified device architecture (CUDA). Yang et al. [17] presented a method to accelerate the computation of molecular dynamics (MD) simulation by GPU, and a speedup factor between 10 and 11 was achieved. The fluid dynamics problem was investigated by the Navier–Stokes solver on a GPU by some researchers [11,18]. Tomov et al. [19] realized their implementation of a Monte Carlo type simulation on a GPU and found that the GPU-based simulation can be two-to-six times faster than that of a CPU for certain applications. Recently, the implementation of the finite element tearing and interconnect (FETI) method to a hybrid CPU/GPU computing environment was reported by Papadrakakis et al., and they concluded that the hybrid CPU/GPU computing platforms had tremendous potential [20]. GPU has been widely used to accelerate computation. However, there are few studies reported concerning the GPU implementation of the lattice Boltzmann method for flows with curved boundaries.

In this paper, in order to efficiently accomplish the simulation of a flow with curved boundaries, a parallel LBM algorithm on a GPU is developed. At the same time, the accuracy of the results and the factors that influence the efficiency of the parallel algorithm are discussed in detail.

The structure of this paper is organized as follows. Section 1 is the introduction. Section 2 briefly describes the lattice Boltzmann method. In Section 3, the parallel computing technology on a GPU developed by nVidia corporation is introduced. Section 4 illustrates the problem and the sets of boundary conditions. The simulation results and the factors that influence the efficiency of the parallel algorithm are presented in Section 5. Finally, conclusions are drawn in Section 6.

* Corresponding author. Tel.: +86 571 87952598; fax: +86 571 87951616.
E-mail address: zhouhao@cmee.zju.edu.cn (H. Zhou).

0045-7825/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cma.2012.03.011

2. Lattice Boltzmann method

Unlike conventional numerical schemes based on the discretization of macroscopic continuum equations, the lattice Boltzmann method is based on microscopic models and mesoscopic kinetic equations [5]. LBM is considered to be a special discretization scheme of the Boltzmann equation [21,22]. In LBM, particle distribution functions (mass fractions) collide and propagate on a regular grid. The model used in this paper is the lattice BGK (LBGK) model from Qian [23], which is the most widely used model, and DnQb (n is the dimension of the space and b is the discretized speed) model proposed by Qian is the most representative. In LBGK model, the evolution of the density distribution f for a single fluid particle is given by:

$$\frac{Df}{Dt} = \partial_t f + \left(\vec{\xi}\cdot\nabla\right)f = \frac{f^{eq}-f}{\tau} \tag{1}$$

where $\vec{\xi}$ is the microscopic velocity, f^{eq} is the Maxwell-Boltzmann equilibrium density distribution function and τ is the relaxation time. The macroscopic variables such as velocity $\vec{u}$ and density ρ can be obtained by:

$$\begin{aligned}\rho(\vec{x},t) &= \int f\left(\vec{x},\vec{\xi},t\right)d\vec{\xi}\\ \rho(\vec{x},t)\vec{u}(\vec{x},t) &= \int \vec{\xi} f\left(\vec{x},\vec{\xi},t\right)d\vec{\xi}\end{aligned} \tag{2}$$

The velocity space should be discretized to obtain the lattice Boltzmann model. Suppose that the distribution function moves along the lattice link $d\vec{x}_i$ with the particle speed $\vec{e}_i$ during dt, it can be concluded that $d\vec{x}_i = \vec{e}_i dt$. The evolution equation can be described as:

$$f_i(\vec{x}+\vec{e}_i dt, t+dt) - f_i(\vec{x},t) = \frac{f_i^{eq}-f_i}{\tau} \tag{3}$$

The macroscopic variables such as velocity $\vec{u}$ and density ρ can then be given by:

$$\begin{aligned}\rho &= \sum_i f_i\\ \rho\vec{u} &= \sum_i \vec{e}_i f_i\end{aligned} \tag{4}$$

In this paper, a nine-velocities two-dimensional (D2Q9) lattice has been used, as shown in Fig. 1. In D2Q9 model, there are three kinds of particle speed, and they are formulated by:

$$\vec{e}_i = \begin{cases}(0,0) & i=0\\ c\left(\cos\frac{(i-1)\pi}{2}, \sin\frac{(i-1)\pi}{2}\right) & i=1,2,3,4\\ \sqrt{2}c\left(\cos\frac{(2i-1)\pi}{2}, \sin\frac{(2i-1)\pi}{2}\right) & i=5,6,7,8\end{cases} \tag{5}$$

where c is the lattice speed. The equilibrium density distribution function is given by

$$f_i^{eq} = \rho\omega_i\left[1+\frac{\vec{e}_i\cdot\vec{u}}{c_s^2}+\frac{(\vec{e}_i\cdot\vec{u})^2}{2c_s^4}-\frac{u^2}{2c_s^2}\right] \tag{6}$$

where c_s is the sonic velocity, which equals to $1/\sqrt{3}$; while ω_i is the weight coefficient, and should satisfy corresponding constraints such as energy conservation, momentum conservation, etc., and it has the following values:

$$\omega_i = \begin{cases}\frac{4}{9} & i=0\\ \frac{1}{9} & i=1,2,3,4\\ \frac{1}{36} & i=5,6,7,8\end{cases} \tag{7}$$

The LBM code, which can be split into collision and propagation steps, is easy to implement. In the collision step, the distribution functions of a certain node will not exchange with its neighbor, and the distribution function after collision is given by:

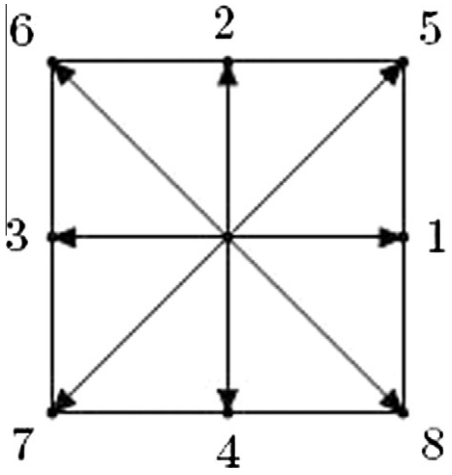

Fig. 1. Schematic diagram of D2Q9 model [6].

$$f_i^*(\vec{x},t) = \frac{f_i^{eq}-f_i}{\tau} \tag{8}$$

The propagation step is:

$$f_i(\vec{x}+\vec{e}_i dt, t+dt) = f_i^*(\vec{x},t) \tag{9}$$

The distribution functions of a certain node will exchange with its neighbors during the propagation step. However, the distribution functions in the nodes of boundaries are unknown, and should be constructed before the propagation step.

There are many methods to deal with the boundary. The non-equilibrium extrapolation scheme developed by Guo [24,25] is used in this study, for its reliable numerical stability and second order accuracy. The main idea of the non-equilibrium extrapolation scheme is that the distribution function of the boundary node is decomposed into equilibrium and non-equilibrium parts:

$$f_i(\vec{x}_b,t) = f_i^{eq}(\vec{x}_b,t) + f_i^{neq}(\vec{x}_b,t) \tag{10}$$

where $\vec{x}_b$ is the boundary node, while $f_i^{eq}(\vec{x}_b,t)$ and $f_i^{neq}(\vec{x}_b,t)$ are the equilibrium and non-equilibrium parts of the distribution function of $\vec{x}_b$. As shown in Section 2, the velocity of the boundary node $\vec{u}_b$ is known, while the density ρ_b is unknown. Thus, the equilibrium part can be obtained approximately by:

$$f_i^{eq}(\vec{x}_b,t) = f_i^{eq}\left(\rho(\vec{x}_f,t),\vec{u}_b\right) \tag{11}$$

where $\vec{x}_f$ is the adjacent fluid node of $\vec{x}_b$, and $\vec{u}_f$ is its velocity. The non-equilibrium part can be obtained approximately by:

$$f_i^{neq}(\vec{x}_b,t) = f_i^{neq}(\vec{x}_f,t) = f_i(\vec{x}_f,t) - f_i^{eq}(\vec{x}_f,t) \tag{12}$$

The curved boundary can also be dealt with by the non-equilibrium extrapolation scheme, as shown in Fig. 2. As for the curved boundary node $\vec{x}_b$, the density can be obtained by $\rho_b = \rho_f$, and the velocity can be obtained by:

$$\begin{cases}\vec{u}_b = \frac{\vec{u}_w+(q-1)\vec{u}_f}{q} & q \geqslant q_c\\ \vec{u}_b = \frac{\vec{u}_w+(q-1)\vec{u}_f}{q} + (1-q)\frac{2\vec{u}_w+(q-1)\vec{u}_{ff}}{1+q} & q < q_c\end{cases} \tag{13}$$

where q_c equals to 0.75, and q is given by:

$$q = \frac{|\vec{x}_f-\vec{x}_b|}{|\vec{x}_f-\vec{x}_w|} \tag{14}$$

And the non-equilibrium part can be obtained by:

$$\begin{cases}f_i^{neq}(\vec{x}_b,t) = f_i(\vec{x}_f,t) - f_i^{eq}(\vec{x}_f,t) & q \geqslant q_c\\ f_i^{neq}(\vec{x}_b,t) = q\left[f_i(\vec{x}_f,t) - f_i^{eq}(\vec{x}_f,t)\right] + (1-q)\left[f_i(\vec{x}_{ff},t) - f_i^{eq}(\vec{x}_{ff},t)\right] & q < q_c\end{cases} \tag{15}$$