The implementation of multi-block lattice Boltzmann method on GPU

***Ya Zhang[1], †Guang Pan[1], and Qiaogao Huang[1]**

[1] School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China.

*Presenting author: zhangya9741@163.com

†Corresponding author: panguang601@163.com

Abstract

A straightforward implementation of multi-block lattice Boltzmann method (MB-LBM) on a graphical processing unit (GPU) is presented to accelerate simulations of complex fluid flows. The characteristics of MB-LBM algorithm are analyzed in detail. The algorithm is tested in terms of accuracy and computational time with the benchmark cases of lid driven cavity flow and the flow past a circular cylinder, and satisfactory results are obtained. The results show the performance on GPU is consistently better than that on CPU, and the greater the amount of data, the larger the acceleration ratio. Moreover, the arrangement of computational domain has significant effects on the performance of GPU. These results demonstrate the great potential of GPU on MB-LBM, especially for the calculation with large amounts of data.

Keywords: Multi-block, Lattice Boltzmann method, Graphical processing unit, Ratio of acceleration.

Introduction

During recent decades, the lattice Boltzmann method (LBM) has developed into an alternative method for simulating complex fluid flow [1]. LBM is based on the statistical physics and originally came from the Boltzmann equation. A direct connection between the lattice Boltzmann equation and Navier-Stokes equations has been established under the nearly incompressible condition [2]. The fact that LBM evolves rather locally makes it more suitable for parallel computing compared to the conventional computation method.

Graphical processing unit (GPU) is designed to process large graphics data sets for rendering tasks, so it has exceeded the computation speed of PC-based central processing unit (CPU) by more than one order of magnitude while being available for a comparable price. Another advantage for GPU application is that Compute Unified Device Architecture (CUDA) provided by NVIDIA, a standard C language extension for parallel application development on a GPU, reduces the development threshold of GPU programming greatly. Due to the inherent parallelism of LBM, a significant speedup of GPU-based computation on LBM has been reported in different areas. Fan et al. [3] implemented the LBM simulations on a cluster of GPUs with message passing interface (MPI). Tolke and Krafczyk [4] implemented a three-dimensional LBM and achieved near teraflop computing on a single workstation. Zhou et al. [5] provided an efficient GPU implementation of flows with curved boundaries, leading to nearly an 18-fold speed increase. Tubbs et al. [6] implemented LBM for solving the shallow water equations and the advection dispersion equation on GPU-based architectures, and the results indicate the promise of the GPU-accelerated LBM for modeling mass transport phenomena in shallow water flows. GPU has tremendous potential to accelerate LBM computation owing to the parallel nature of LBM.

The traditional LBM is often employed on uniform grids, which makes the evolution explicit and the algorithm simple, but at the same time could increase the computational effort dramatically on the road to high resolution. To solve this problem, a multi-block lattice Boltzmann method (MB-LBM) is designed and applied over the flow area where relatively high resolution is needed. As a useful tool of grid refinements in LBM, the multi-block technique has been investigated in recent years. In 1998, Filippova et al. [7] introduced a local second order refinement scheme and provided the theoretical foundation for multi-block techniques. In 2000, Lin and Lai [8] designed a composite block-structured scheme by placing the fine grid blocks on needed area for the mesh refinement. In 2002, Yu et al. [9] proposed a multi-block scheme, where the fine block is partially overlapped at the interfacial lattices, increasing the model efficiency greatly. The model has been successfully applied to various areas. Yu and Girimaji [10] extended this model to 3D turbulence simulations. Y. Peng et al. [11] applied it in the immersed boundary lattice Boltzmann method with multi-relaxation-time collision scheme. Liu et al. [12] validated the multi-block lattice Boltzmann model coupled with the large eddy simulation model in transient shallow water flows simulation. Farhat et al. [13][14] extended the single phase MB-LBM to the multiphase Gunstensen model, in which the grid was free to migrate with the suspended phase, and validated a 3D migrating multi-block model. Following from this, the present study aims to develop an efficient and straightforward algorithm for the GPU implementation of MB-LBM, and test it in terms of accuracy and computational time.

Multi-block lattice Boltzmann method

In the present study, the BGK lattice Boltzmann method is used with a two-dimensional nine-velocity (D2Q9) discrete velocity model [2], as shown in Fig. 1. The lattice Boltzmann method formulates as the following evolution equation:

$$f_\alpha(\boldsymbol{x}+\boldsymbol{e}_\alpha\delta t, t+\delta t) = f_\alpha(\boldsymbol{x},t) - \frac{1}{\tau}\left[f_\alpha(\boldsymbol{x},t) - f_\alpha^{eq}(\boldsymbol{x},t)\right] \tag{1}$$

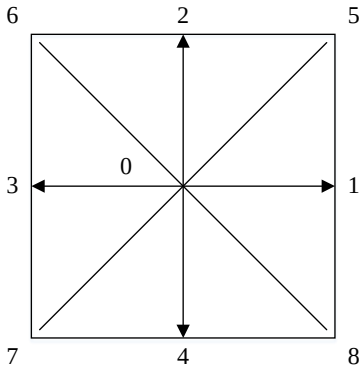


Figure 1. Lattice pattern: D2Q9

where f_α is the particle distribution functions representing the probability of particles at position $\boldsymbol{x}$ and discrete velocity $\boldsymbol{e}_\alpha$ at time t; δt is the time step; τ is the single-relaxation-time,

depending on the kinematic viscosity ν, $\tau = 3\nu + 0.5$; e_α is the α th discrete velocity, the discrete velocity model is

$$\boldsymbol{e} = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix} \tag{2}$$

f_α^{eq}, the approximate of the Maxwell-Boltzmann equilibrium distribution function at low numbers, is expressed as follow:

$$f_\alpha^{eq} = \rho w_i \left[1 + \frac{\boldsymbol{e}_\alpha \cdot \boldsymbol{u}}{c_s^{\,2}} + \frac{(\boldsymbol{e}_\alpha \cdot \boldsymbol{u})^2}{2c_s^{\,4}} - \frac{u^2}{2c_s^{\,2}} \right] \tag{3}$$

where w_α is the weighting coefficient, valued by $w_0 = 4/9$, $w_1 = w_2 = w_3 = w_4 = 1/9$ and $w_5 = w_6 = w_7 = w_8 = 1/36$; the sound speed is $c_s = 1/\sqrt{3}$; ρ and $\boldsymbol{u}$ are the macroscopic density and velocity, which can be calculated from the distribution function respectively by:

$$\rho = \sum_{\alpha=0}^{8} f_\alpha = \sum_{\alpha=0}^{8} f_\alpha^{eq} \tag{4-1}$$

$$\rho \boldsymbol{u} = \sum_{\alpha=0}^{8} \boldsymbol{e}_\alpha f_\alpha = \sum_{\alpha=0}^{8} \boldsymbol{e}_\alpha f_\alpha^{eq} \tag{4-2}$$

This paper uses the multi-block method proposed by Yu et al. [9], which satisfies the continuity of mass, momentum and stresses across the interface. To illustrate the basic idea, a two-block system consisting of a coarse block and a fine block is shown in Fig. 2.

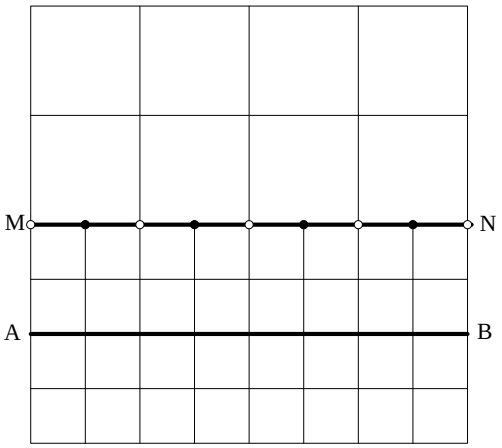


Figure 2. Interfaces structure between two blocks

The ratio of the lattice space between coarse blocks and fine blocks *m* is defined as:

$$m = \delta x_c / \delta x_f = m_c m_f \tag{5}$$

where the subscript *c* refers to the coarse block while *f* refers to the fine block, δx_c and δx_f are the lattice space, $m_c = 1$ and $m_f = \delta x_c / \delta x_f$ are the lattice space parameters. To maintain a consistent viscosity across blocks, the relaxation time τ_f on the fine block and τ_c on the coarse block have to satisfy the following equation:

$$\tau_f = 0.5 + m_f(\tau_c - 0.5) \tag{6}$$

The transfer of the post-collision distribution functions between different blocks happens after the collision step. Since each interface grid consists of overlapping two sets of coarse and fine nodes, the information of coarse boundary nodes can be obtained after m_f steps of evolution on the fine grid, where the post-collision distribution $\tilde{f}_\alpha^c$ for the coarse block is written as:

$$\tilde{f}_\alpha^c = f_\alpha^{eq,f} + m_f \frac{\tau_c - 1}{\tau_f - 1}(\tilde{f}_\alpha^f - f_\alpha^{eq,f}) \tag{7}$$

Similarly, when transferring the data from the coarse block to the fine block, one follows:

$$\tilde{f}_\alpha^f = f_\alpha^{eq,c} + \frac{\tau_f - 1}{m_f(\tau_c - 1)}(\tilde{f}_\alpha^c - f_\alpha^{eq,c}) \tag{8}$$

As shown in Fig. 2, the line MN is the fine block boundary, while the line AB is the coarse block boundary. The information on the nodes noted by solid symbol can be obtained through spatial interpolation based on the information at the open nodes on the line MN.

To eliminate the possibility of spatial asymmetry caused by interpolations, a symmetric cubic spline fitting is used to calculate the unknown nodes on the fine blocks [9], which is done by

$$\begin{aligned} &\tilde{f}(x) = a_i(x_i - x)^3 + b_i(x - x_{i-1})^3 + c_i(x_i - x) + d_i(x - x_{i-1}) \\ &x_{i-1} \le x \le x_{i+1} \end{aligned} \tag{9}$$

where according to the continuity of the nodal condition of $\tilde{f}$ and $\tilde{f}'$ (the first order derivation of $\tilde{f}$), and suitable end condition, the coefficients (a_i, b_i, c_i, d_i) in Eq. (9) are computed as follows:

$$
\begin{aligned}
a_i &= \frac{M_{i-1}}{6h_i} \\
b_i &= \frac{M_i}{6h_i} \\
c_i &= \frac{\tilde{f}_{i-1}}{h_i} - \frac{M_{i-1}h_i}{6} \\
d_i &= \frac{\tilde{f}_i}{h_i} - \frac{M_i h_i}{6}
\end{aligned}
\tag{10}
$$

where M_i is the second order derivatives of $\tilde{f}_i$, following the equation

$$
0.5M_{i-1} + 2M_i + 0.5M_{i+1} = 3(2f_i - f_{i-1} - f_{i+1}) \tag{11}
$$

The natural spline end condition is stipulated with $M_0 = M_n = 0$.

A three-point Lagrangian scheme is used in the temporal interpolation of the post-collision distribution function on the interface grid at the specific time:

$$
\tilde{f}_i^{\,f}(t) = \sum_{k=-1}^{1} \tilde{f}_i^{\,f}(t_k) \left(\prod_{\substack{k'=-1 \\ k \neq k'}}^{1} \frac{t - t_{k'}}{t_k - t_{k'}} \right) \tag{12}
$$

So the function for the *n*th evolution of the fine block is expressed as

$$
\tilde{f}_i^{\,f}(t) = 0.5\frac{n}{m_f}\left(\frac{n}{m_f} - 1\right)\tilde{f}_i^{\,f}(t_{-1}) - \left(\frac{n}{m_f} - 1\right)\left(\frac{n}{m_f} + 1\right)\tilde{f}_i^{\,f}(t_0) + 0.5\frac{n}{m_f}\left(\frac{n}{m_f} + 1\right)\tilde{f}_i^{\,f}(t_1) \tag{13}
$$

where the present time is $t = t_0 + \frac{n}{m_f}$.

The flow chart of the computational sequence for the MB-LBM in the two-block system is shown in Fig. 3.

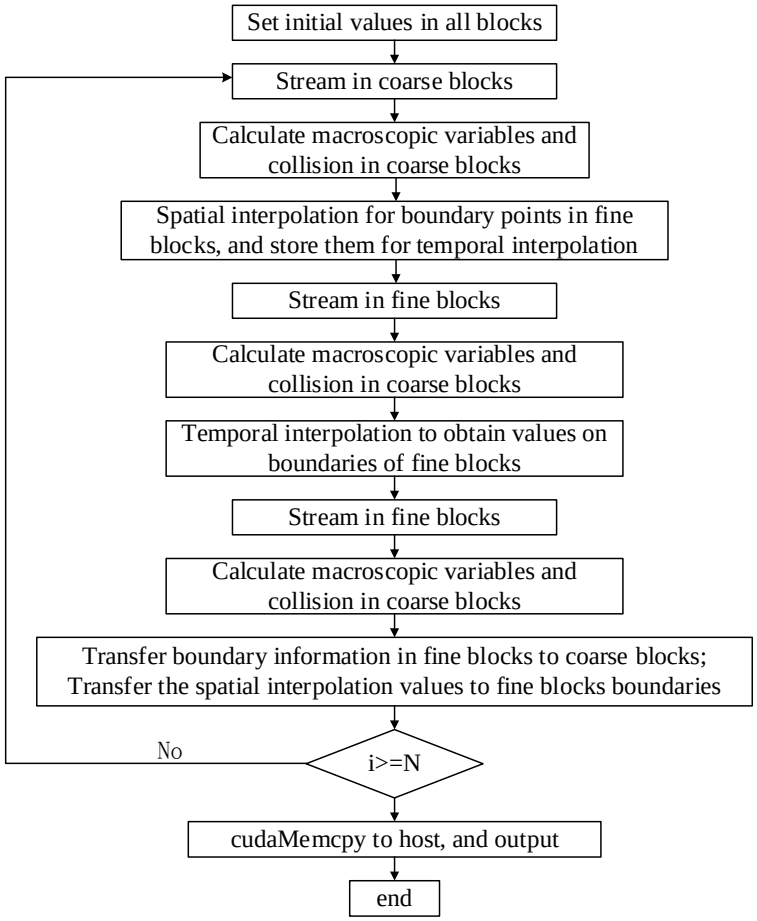


Figure 3. The flow chart of the computational sequence for MB-LBM

In this paper the momentum-exchange method [15] is used to calculate the force exerted onto the obstacle considering its simplicity. In order to differentiate the nodes in the computational domain, node type is employed to donate the fluid node, boundary node of computational domain, boundary node of blocks and solid node. If particles in the solid node $\boldsymbol{x}_b(i, j)$ of the fine block, will move to a fluid node along the direction $\boldsymbol{e}_\alpha$ in the next step, values (i, j, α) should be stored in an array. The force can be calculated by

$$\boldsymbol{F} = \frac{1}{m} \sum_{All\ (i,j,\alpha)} \left[\boldsymbol{e}_\alpha f_\alpha(\boldsymbol{x}_b) - \boldsymbol{e}_{\bar{\alpha}} f_{\bar{\alpha}}(\boldsymbol{x}_b + \boldsymbol{e}_\alpha \delta t) \right] \tag{14}$$

GPU implementation

A graphical processing unit (GPU) is specifically designed to process large graphics data sets for rendering tasks. As GPU has a number of processing cores, so besides graphic rendering tasks, it also is used to implement other parallel computing tasks. In this work, the simulation is carried out on a CPU platform of Intel Xeon(R) W3550, 3.07GHz) with RAM of 24.0 GB and a NVIDIA GPUs device (Geforce 980ti), programming using CUDA (Compute Unified Device Architecture).

In the CUDA programing architecture, CPU is considered to be the host, while GPU is considered

to be the device. The code is split up into a CPU and GPU part, the latter is called kernel, compiled by NVIDIA C-Compiler (NVCC). When a kernel function is launched with required parameters, the number of blocks and the number of threads in each block (256 in this paper), it is executed by these threads on a device. In one block, each thread is indexed by a thread identification. Threads from different blocks cannot communicate, while threads from the same block are independent, but can communicate via shared memory and have synchronize execution. A kernel is executed in a grid of thread blocks indexed by a block identification. The grid terminates when all threads of a kernel complete their execution, and the execution continues on the host until another kernel is launched.

The memory access of the kernel has a great influence on the implementation performance. The registers are trace buffer on GPU, and can be accessed with nearly no time delay, but is rather small, so excessive local variables used in kernel should be avoided. The global memory is a device memory and is the largest memory device in GPU, but not as fast as the registers. In this work, each node requires nearly 200 bytes of memory for double precision computation, so most of the data will be stored in the global memory. Besides, there is a share memory for each multiprocessor, allowing communication between threads, and can be accessed as fast as the registers. The constant memory, which can also be fast accessed, is used to store the constants that are read only and are accessed frequently.

The LBM code is highly parallelizable since it can be separated into two main steps, streaming and collision [2]. In the collision step, the distribution functions of a certain node will not exchange with its neighbor, and the post-collision function is given by

$$\tilde{f}_\alpha(\boldsymbol{x},t) = f_\alpha(\boldsymbol{x},t) - \frac{1}{\tau}\left[f_\alpha(\boldsymbol{x},t) - f_\alpha^{eq}(\boldsymbol{x},t) \right] \tag{15}$$

The streaming step is related to the distribution functions of the surrounding nodes according to Eq. (1) and Eq. (15). Considering the fact that misaligned read is faster than misaligned write[16], the streaming is carried out with the following equation

$$f_\alpha(\boldsymbol{x}, t+\delta t) = \tilde{f}_\alpha(\boldsymbol{x} - \boldsymbol{e}_\alpha \delta t, t) \tag{16}$$

To increases the efficiency of data communication, the collision and the streaming step are combined into one kernel to avoid repeated access of global memory for distribution functions.

For systems containing multi-level blocks, according to the flow chart in Fig. 3, the computation can be expressed with a recursive function shown in Fig. 4.

```
void evolution(int level)
{
	for (int i=0; i<m[level]; i++)
	{
		if (level == LEVEL)
			return;
		if (!(level == 0 || i == 0))
		{
			//temporal interpolation
		}
		//stream, calculate macroscopic variables and collision
		//information exchange between the present level blocks
		if (level != LEVEL-1)
		{
			//spatial interpolation to prepare for blocks, level+1
		}
		evolution(level+1);
		if (level != 0 && i == m[level]-1)
		{
			//Transfer boundary information in blocks level+1 to level;
			//Transfer the spatial interpolation values to level+1
		}
	}
}
```

Figure 4. Program of the recursive function for MB-LBM

Since there are always the same data types of variables needed to be record in each node, a struct body, including pointers to node type, position, density, velocity, distribution functions and post-collision distribution functions, is created to store variable information. With these pointers, memory in host and device is allocated dynamics for the variables.

In the stage of the spatial interpolation, it is needed to obtain M_i in Eq. (10) and Eq. (11). In serial processing, the tridiagonal matrix in Eq. (11) is solved with the Thomas algorithm, which is almost unfeasible in parallel algorithm. The cuSPARSE library presented by NVIDIA contains a set of basic linear algebra subroutines used for handling sparse matrices in parallel mode. The function cusparseDgtsv() is employed in this paper. It can be used by cusparseDgtsv(cusparseHandle_t *handle*, int *m*, int *n*, const double **dl*, double **d*, double **du*, double **B*, int *ldb*), where *handle* is the handle to the cuSPARSE library context; *m* is the size of the linear system (must be larger than or equal to three); *n* is the columns of matrix *B*, which means M_i for different variables can be solve in a single call; array *dl*, *d*, *du* contain the lower, the main, the upper diagonal of the tridiagonal linear system, respectively; *B* is the right-hand-side array, *ldb* is the leading dimension of *B*. The solution will be written in array *B* before the function completes.

It is obvious that the spatial interpolation in parallel is much more complex than the temporal interpolation, so it is suggested that the largest ratio of the lattice space between adjacent levels should be placed on the finest level. And in this work, the arrangement of levels is expressed in form of m_1-…-m_i-…-m_n in coarse-fine order, where m_1 is always 1, m_i is the ratio of the lattice space of level *i* to that of level *i*-1. So as mentioned, the arrangement of levels 1-2-3 is better than 1-3-2.

In this work, all the procedures but output are completed on the GPU directly to eliminate the unnecessary copy between host and device. At the same time due to the fact that the atom function atomicAdd() in the CUDA toolkit provided by NVIDIA can only be used for Integer and Long, the parallel reduction is used to calculate the force in Eq. (14) after loading the position and direction.

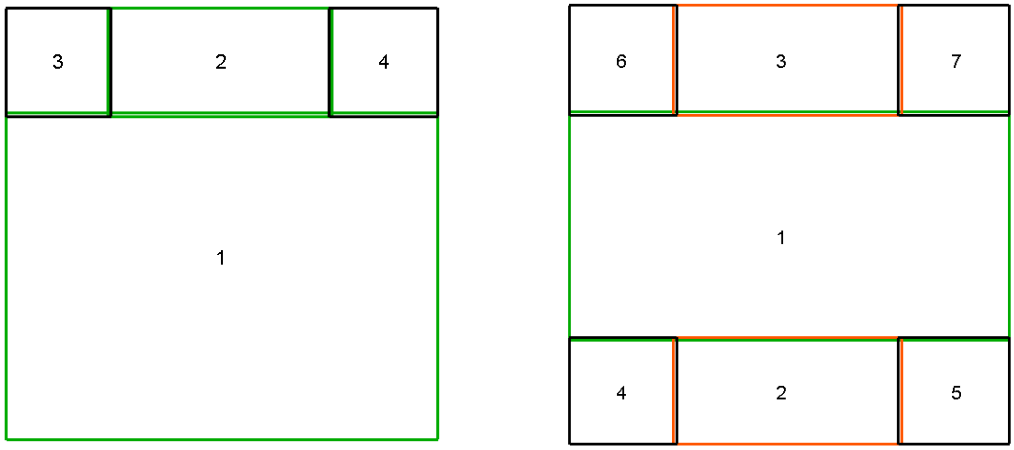
Presentation of test cases and discussion

Lid driven cavity flow

The lid driven cavity flow has been extensively used as a benchmark problem to test the accuracy of a numerical method. The computations are carried out using the multi-block computational domains, whose schematic diagrams are shown in Fig. 5.

In all the arrangements, the finest blocks are placed on the areas of singularity points or changing sharply. As shown in Fig. 5, the finest blocks is located in the two upper corner regions. In Fig. 5(a), there are two levels of blocks and four separate blocks in the calculation. Block 1 and block 2 belong to the first level; block 3 and block 4 belong to the second level; the diagram in Fig. 5(b) contains three levels and seven blocks, while block 1 belong to the first level, block 2 and block 3 belong to the second level, and block 4 to block 7 belong to the third level.

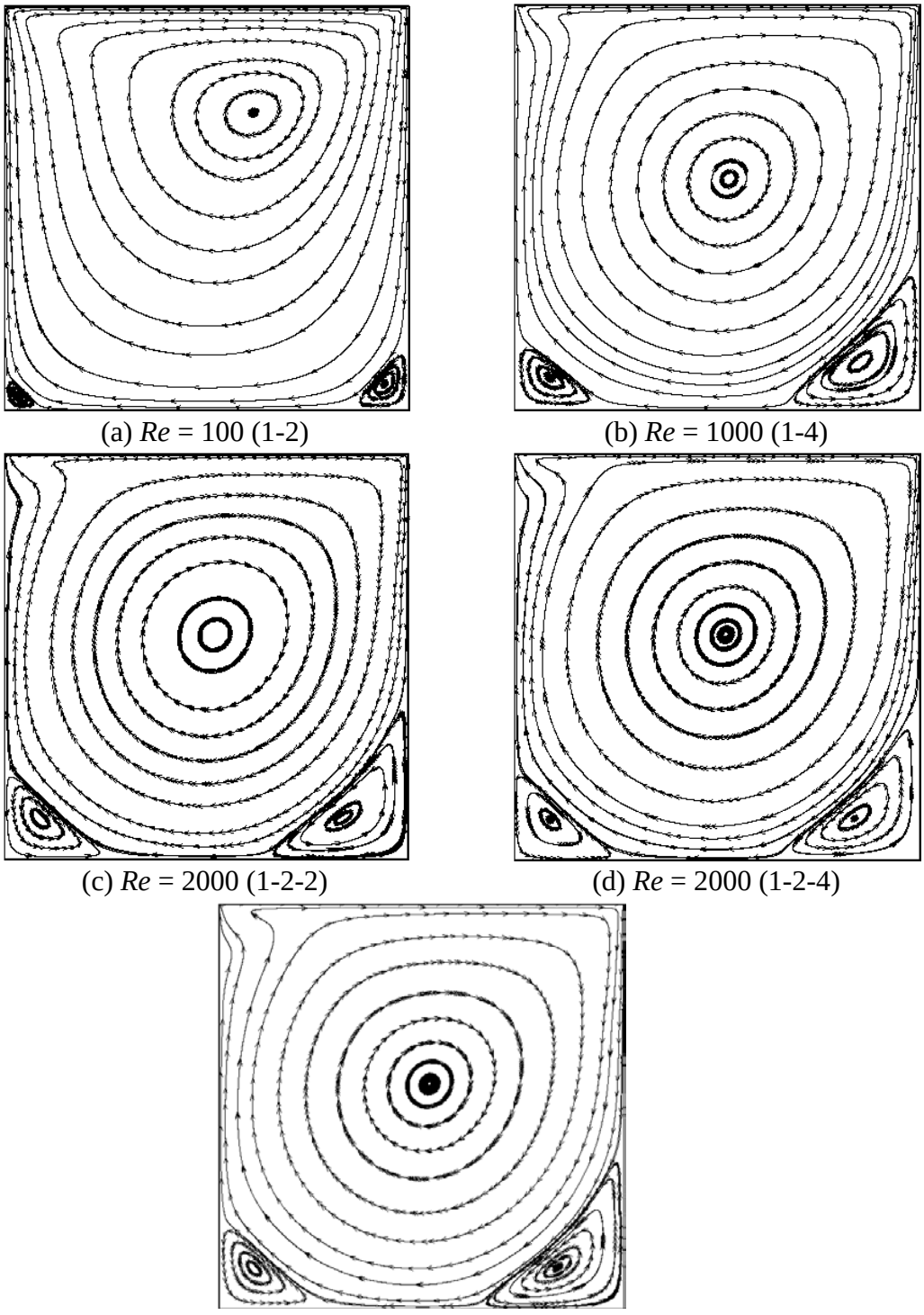
The simulation region is 128-128. The initial condition for density is unity and that for velocity is zero. The upper wall velocity is U = 0.1. All the boundaries uses the moving boundary half-way bounce-back scheme.



(a) (b)

Figure 5. Arrangements of blocks for the lid driven cavity flow

To assess the results, the solutions of Ref. [17] and Ref. [18] are used for comparison. The dimensionless locations of the centers of the primary vortex, the lower left vortex and the lower right vortex of present work and of previous literatures are listed in Table 1. As shown in Table 1, all the results show a good agreement with previous researches. And for Re = 2000, different arrangements of blocks appear identical results.



(a) *Re* = 100 (1-2)

(b) *Re* = 1000 (1-4)

(c) *Re* = 2000 (1-2-2)

(d) *Re* = 2000 (1-2-4)

(e) *Re* = 2000 (1-4-2)

Figure 6. Streamlines for the lid driven flow

Table 1 Comparison of the vortex centers with previous literatures [17][18]

Re	Arrangement	Primary vortex	Lower left vortex	Lower right vortex
100				
Present	1-2 (Fig. 5(a))	(0.6142, 0.7402)	(0.0354, 0.0394)	(0.9370, 0.0669)
Ref. [17]		(0.6172, 0.7344)	(0.0313, 0.0391)	(0.9453, 0.0625)

1000				
Present	1-4 (Fig. 5(a))	(0.5276, 0.5669)	(0.0866, 0.0787)	(0.8504, 0.1181)
Ref. [17]		(0.5313, 0.5625)	(0.0859, 0.0781)	(0.8594, 0.1094)
2000				
Present	1-2-2 (Fig. 5(b))	(0.5238, 0.5555)	(0.0873, 0.1032)	(0.8413, 0.0992)
	1-2-4 (Fig. 5(b))	(0.5238, 0.5555)	(0.0873, 0.1032)	(0.8413, 0.0992)
	1-4-2 (Fig. 5(b))	(0.5238, 0.5555)	(0.0873, 0.1032)	(0.8413, 0.0992)
Ref. [18]		(0.5250, 0.5500)	(0.0875, 0.1063)	(0.8375, 0.0938)

Flow past a circular cylinder

A flow past a circular cylinder is simulated to implement the parallel algorithm in simulation domain that has more levels and blocks.

The arrangement of the computational domain is shown in Fig. 7. There are four levels of blocks in the simulation. Block 1 to block 4 belong to the first level; block 5 and block 6 belong to level two; block 7 to block 9 belong to level 3; block 10 belong to level 4, the finest level. The ratio of the lattice space between adjacent levels is 1-2-2-2.

In this calculation, the cylinder diameter D is set to 6. The length of the simulation region is 320, and the width is 128. The center of the cylinder is at (64, 64), which makes it located in the finest block, as shown in Fig. 7. The slip boundary scheme is implemented on the top and bottom boundaries. The standard bounce back scheme is used on the cylinder surface. The velocity and the pressure scheme of Zou and He are applied on the inlet and the outlet boundaries, respectively, where the far field velocity is U_0=0.1 and the initial density is unity. The relaxation time for the first level grid is computed by Re=100, based on the far field velocity and the diameter of the cylinder.

Drag coefficient, lift coefficient and Strouhal number are the benchmark dimensionless numbers for the flow past a circular cylinder. The drag and the lift coefficients are calculated using the following formulae, $C_D = \frac{2F_D}{\rho U^2 D}$ and $C_L = \frac{2F_L}{\rho U^2 D}$, and the Strouhal number is defined as $St = \frac{aD}{U}$, where F_L the lift force, F_D the drag force, D the cylinder diameter, a the frequency of vortex-shedding, obtained by processing F_L with Fast Fourier Transform.

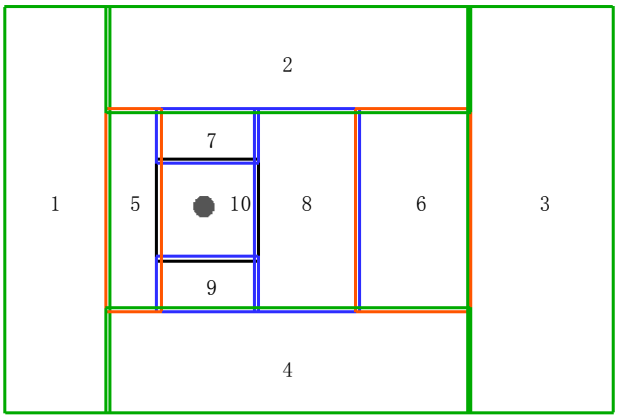


Figure 7. Arrangement of blocks for the flow past a circular cylinder

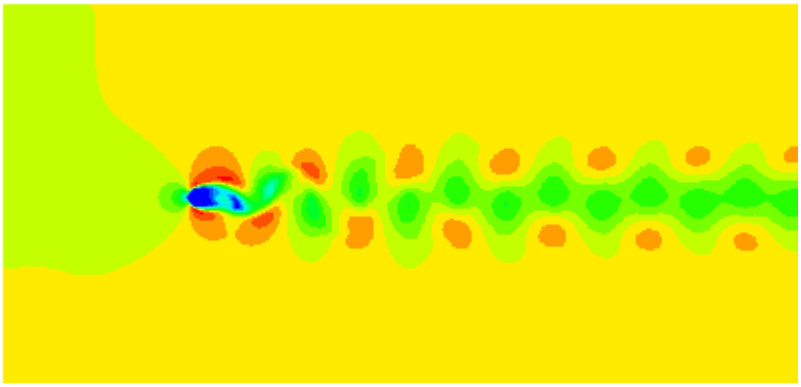


Figure 8. Velocity contour for the flow past a circular cylinder

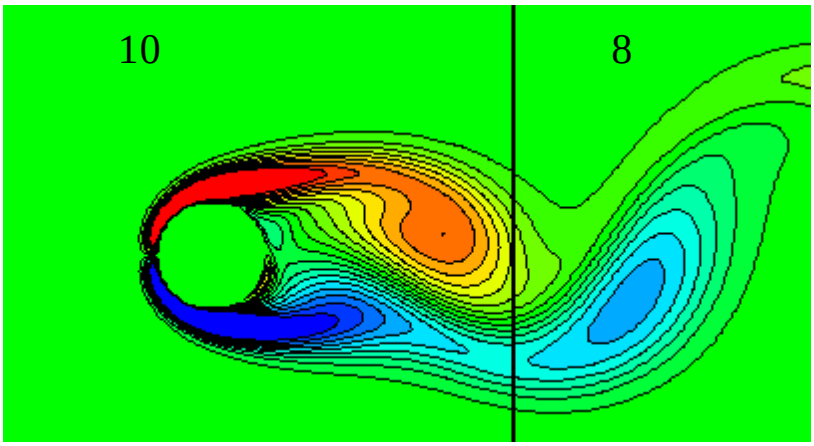


Figure 9. Vorticity contour for the flow past a circular cylinder

Table 3 Comparison of results at Re = 100 with previous literatures [19][20][21]

Author	C_D	C_L	S_t
Silva [19]	1.39	-	0.16
Zhou [20]	1.428	0.315	0.172
Xu [21]	1.423	0.34	0.171
This work	1.381	0.304	0.168

The velocity contour for the flow past a circular cylinder is shown in Fig. 8. The instantaneous vorticity contours of vortex shedding are plotted in Fig. 9. It can be seen clearly that the vorticity is rather smooth across the block interface. This shows that the implementation of multi-block scheme functions well for unsteady flow. Table 3 shows our numerical results compare well with the previous results, despite little differences.

Assess the performance of MB-LBM code on GPU

The parameters of performance of MB-LBM on CPU and on GPU is shown in Table 2, including the time spending for evolution of 10^4 steps (in second), the number of lattice updates per step in an arrangement (LUPS), million lattice updates per second (MLUPS), and the acceleration ratio of GPU to CPU. In general, LUPS represents the amount of data, and a large MLUPS means a high data processing speed.

Table 2 Performance of CPU and GPU for 104 steps

Case	Arrangement	LUPS	CPU		GPU		Acceleration ratio
			Time	MLUPS	Time	MLUPS	
1	1-2 (Fig. 5(a))	31267	205.66	1.52	64.52	4.85	3.19
2	1-4 (Fig. 5(a))	145691	1083.28	1.34	86.63	16.82	12.50
3	1-2-2 (Fig. 5(b))	300688	1894.23	1.59	245.59	12.24	7.71
4	1-2-4 (Fig. 5(b))	2090912	19543.09	1.07	498.00	42.00	39.24
5	1-4-2 (Fig. 5(b))	2326104	21736.04	1.07	659.12	35.29	33.00
6	1-2-2-2 (Fig. 7)	790392	5835.10	1.35	578.71	13.66	10.08

It can be seen from Table 2 that the ratio of acceleration is not a constant, and performance on GPU is always better than that of CPU. To be specifically, as the amount of data increases, roughly the speedup is more obvious. Besides, the arrangement of computational domain has great impact on the performance of GPU. In case 2 and case 3, the resolution of upper corners is the same, but on GPU the performance of case 2 is much better while with a smaller LUPS, so it is not recommended to employ more levels for the same resolution. In addition, according to the performance of case 4 and case 5, considering the time consumed by spatial interpolation in MB-LBM, it is verified that the largest ratio of the lattice space between adjacent levels should be placed on the finest level.

Conclusion

In this paper, a straightforward multi-block LBM parallel algorithm based on a single GPU has been presented. The characteristics of MB-LBM algorithm are analyzed in detail. The benchmark cases of the lid driven cavity flow and the flow past a circular cylinder are investigated as the test cases for the GPU-based implementation, and satisfactory results are obtained. Performance on GPU is always better than that of CPU, and the greater the amount of data, the larger the acceleration ratio. And arrangement of computational domain has significant effects on the performance. The largest acceleration ratio 39.24 are achieved by now, however that still leaves room for a large rise in computation with large amounts of data.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (11502210, 51279165).

References

[1] Aidun, C. K. and Clausen, J. R. (2010) Lattice-Boltzmann method for complex flows, *Annual review of fluid mechanics* **42**, 439-472.
[2] Mohamad, A. A. (2011) Lattice Boltzmann method: fundamentals and engineering applications with computer codes, Springer-Verlag, London, UK.
[3] Fan, Z., Qiu, F., Kaufman, A. and Yoakum-Stover, S. (2004) GPU cluster for high performance computing, *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, **47**.
[4] Tölke, J. and Krafczyk, M. (2008) TeraFLOP computing on a desktop PC with GPUs for 3D CFD, *International Journal of Computational Fluid Dynamics* **22**, 443-456.
[5] Zhou, H., Mo, G., Wu, F., Zhao, J., Rui, M. and Cen, K. (2012) GPU implementation of lattice Boltzmann method for flows with curved boundaries, *Computer Methods in Applied Mechanics and Engineering* **225**, 65-73.
[6] Tubbs, K. R. and Tsai, F. T. C. (2011) GPU accelerated lattice Boltzmann model for shallow water flow and mass transport, *International Journal for Numerical Methods in Engineering* **86**, 316-334.
[7] Filippova, O. and Hänel, D. (1998) Grid refinement for lattice-BGK models, *Journal of Computational Physics* **147**, 219-228.
[8] Lin, C. L. and Lai, Y. G. (2000) Lattice Boltzmann method on composite grids, *Physical Review E* **62**, 2219-2225.
[9] Yu, D., Mei, R. and Shyy, W. (2002) A multi‐block lattice Boltzmann method for viscous fluid flows, *International journal for numerical methods in fluids* **39**, 99-120.
[10] Yu, D. and Girimaji, S. S. (2006) Multi-block lattice Boltzmann method: extension to 3D and validation in turbulence, *Physica A: Statistical Mechanics and its Applications* **362**, 118-124.
[11] Peng, Y., Shu, C., Chew, Y. T. Niu, X. D. and Lu, X. Y. (2006) Application of multi-block approach in the immersed boundary–lattice Boltzmann method for viscous fluid flows, *Journal of Computational Physics* **218**, 460-478.
[12] Liu, H., Zhou, J. G. and Burrows, R. (2010) Lattice Boltzmann simulations of the transient shallow water flows, *Advances in Water Resources* **33**, 387-396.
[13] Farhat, H. and Lee, J. S. Fundamentals of migrating multi-block lattice Boltzmann model for immiscible mixtures in 2D geometries, *International Journal of Multiphase Flow* **36**, 769-779.
[14] Farhat, H., Choi, W. and Lee, J. S. (2010) Migrating multi-block lattice Boltzmann model for immiscible mixtures: 3D algorithm development and validation, *Computers & Fluids* **39**, 1284-1295.
[15] Mei, R., Shyy, W. and Yu, D. and Luo, S. L. (1999) Force Evaluation in the Lattice Boltzmann Method, *APS Division of Fluid Dynamics Meeting Abstracts* **1**.
[16] 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**, 3628-3638.
[17] Ghia, U., Ghia, K. N. and Shin, C. T. (1982) High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, *Journal of computational physics* **48**, 387-411.
[18] Vanka, S. P. (1986) Block-implicit multigrid solution of Navier-Stokes equations in primitive variables, *Journal of Computational Physics* **65**, 138-158.
[19] Silva, A. L. E., Silveira-Neto, A. and Damasceno, J. J. R. (2003) Numerical simulation of two-dimensional flows over a circular cylinder using the immersed boundary method, *Journal of Computational Physics* **189**, 351-370.
[20] Zhou, H., Mo, G., Wu, F., Zhao, J., Rui, M. and Cen, K. GPU implementation of lattice Boltzmann method for flows with curved boundaries, *Computer Methods in Applied Mechanics and Engineering* **225**, 65-73.
[21] Xu, S. and Wang, Z. J. (2006) An immersed interface method for simulating the interaction of a fluid with moving boundaries, *Journal of Computational Physics* **216**, 454-493.