Advances in Applied Mathematics and Mechanics
Adv. Appl. Math. Mech., Vol. **7**, No. 1, pp. 1-12

DOI: 10.4208/aamm.2014.m468
February 2015

Implementation of Multi-GPU Based Lattice Boltzmann Method for Flow Through Porous Media

Changsheng Huang, Baochang Shi*, Nanzhong He and Zhenhua Chai

School of Mathematics and Statistics, Huazhong University of Science and Technology, Wuhan 430074, China

Received 21 January 2014; Accepted (in revised version) 26 June 2014

Abstract. The lattice Boltzmann method (LBM) can gain a great amount of performance benefit by taking advantage of graphics processing unit (GPU) computing, and thus, the GPU, or multi-GPU based LBM can be considered as a promising and competent candidate in the study of large-scale fluid flows. However, the multi-GPU based lattice Boltzmann algorithm has not been studied extensively, especially for simulations of flow in complex geometries. In this paper, through coupling with the message passing interface (MPI) technique, we present an implementation of multi-GPU based LBM for fluid flow through porous media as well as some optimization strategies based on the data structure and layout, which can apparently reduce memory access and completely hide the communication time consumption. Then the performance of the algorithm is tested on a one-node cluster equipped with four Tesla C1060 GPU cards where up to 1732 MFLUPS is achieved for the Poiseuille flow and a nearly linear speedup with the number of GPUs is also observed.

AMS subject classifications: 65Y05, 76M28

Key words: Lattice Boltzmann method, GPU computing, CUDA, porous media, MPI.

1 Introduction

Recently, GPU (graphics processing unit) computing is becoming rapidly popular, especially after the emergence of new programming languages like compute unified device architecture (CUDA) [1]. This is mainly due to the tremendous computing power and superior memory bandwidth of the GPU architecture. For this reason, the GPU computing has also been viewed as a cost-efficient tool to overcome the performance bottleneck. Up

*Corresponding author.
Email: hcshust@163.com (C. Huang), shibc@hust.edu.cn (B. Shi), nzhe@hust.edu.cn (N. He), hustczh@hust.edu.cn (Z. Chai)

 ©2015 Global Science Press

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

to now, many applications have achieved a significant speedup by exploiting the GPU processing power.

The lattice Boltzmann method (LBM) [2, 3] is one of such algorithms that is perfectly suitable for running on the GPU. As a kinetic based approach in modeling fluid flows, the lattice Boltzmann method has been successfully applied in various fields. In order to achieve a desirable performance, several researchers implemented the LBM on a single GPU [4–7], and reported some satisfactory accelerations.

To gain a higher performance for complex problems with a large scale, the multi-GPU based LBM has been developed, and studied preliminarily in some previous works [8–14]. In these works, although almost a linear speedup was achieved, only the GPU based LBM for fluid flow in a simple geometry is investigated. Recently, Bernaschi et al. implemented multi-GPU based lattice Boltzmann method for flows in complex geometries by utilizing sparse lattice representation technique [15, 16], which only stores fluid nodes and could achieve high performance when the volume fraction of the fluid phase is quite small. However, for porous media with a high porosity, the full matrix mode is a better choice and can achieve a higher performance than the sparse matrix mode. In this paper, based on the full matrix mode, we present an alternative multi-GPU based LBM algorithm for fluid flows through porous media, in which message passing interface (MPI) technique is adopted for GPU management.

The rest of this paper is organized as follows. In Section 2, we will present a brief overview on the lattice Boltzmann method. In Section 3, an improved multi-GPU based lattice Boltzmann algorithm is proposed, and then, a detailed analysis on the performance of the algorithm is given in Section 4. In the next Section, as an application, the present algorithm is used to predict permeabilities of three types of porous medium, and finally, some conclusions are summarized in Section 6.

2 The lattice Boltzmann method

The lattice Boltzmann method can be viewed as a discrete scheme of the continuous Boltzmann equation. For every time step, particles collide at each node and then propagate to neighboring sites along discrete directions. A popular class of lattice Boltzmann models is the so-called lattice Bhatnagar-Gross-Krook (BGK) model with a single relaxation time approximation. In this model, the evolution equation reads

$$f_i(\mathbf{x}+\mathbf{c}_i\delta t,t+\delta t)-f_i(\mathbf{x},t)=-\frac{1}{\tau}\left[f_i(\mathbf{x},t)-f_i^{(eq)}(\mathbf{x},t)\right], \tag{2.1}$$

where τ is the dimensionless relaxation time, $f_i(\mathbf{x},t)$ is the density distribution function for the particle moving with velocity $\mathbf{c}_i$ at position $\mathbf{x}$ and time t, $f_i^{(eq)}(\mathbf{x},t)$ is the local equilibrium distribution function. In this paper, a three-dimensional lattice Boltzmann model with nineteen velocities (D3Q19 model) [17] is used, and the corresponding equilibrium

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

distribution function is defined as

$$f_i^{(eq)}(\mathbf{x},t)=\omega_i\rho\left[1+\frac{\mathbf{c}_i\cdot\mathbf{u}}{c_s^2}+\frac{(\mathbf{c}_i\cdot\mathbf{u})^2}{2c_s^4}-\frac{|\mathbf{u}^2|}{2c_s^2}\right], \tag{2.2}$$

where ω_i is the weighting factor, and given by $\omega_0=1/3$, $\omega_{1-6}=1/18$, $\omega_{7-18}=1/36$, ρ, $\mathbf{u}$ are the fluid density and velocity, c_s is the sound speed. The discrete velocities $\mathbf{c}_i$ in Eq. (2.2) are defined by

$$\mathbf{c}_i=c\mathbf{e}_i=c\begin{cases}(0,0,0), & i=0,\\ (\pm1,0,0),(0,\pm1,0),(0,0,\pm1), & i=1-6,\\ (\pm1,\pm1,0),(\pm1,0,\pm1,),(0,\pm1,\pm1), & i=7-18,\end{cases} \tag{2.3}$$

where $c=\delta x/\delta t$, δx and δt are the lattice spacing and time step, respectively. The relation between c_s and c can be expressed as $c_s=c/\sqrt{3}$. The macroscopic density and velocity are computed by

$$\rho(\mathbf{x},t)=\sum_{i=0}^{18}f_i(\mathbf{x},t),\quad \rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)=\sum_{i=0}^{18}\mathbf{c}_i f_i(\mathbf{x},t). \tag{2.4}$$

For fluid flows in complex geometries, the most widely used boundary condition is the halfway bounce-back scheme [18], which means particles that hit the boundary, and then simply return back in the direction where they came from. In our simulations, whether to apply the halfway bounce-back scheme or do a normal propagation, it depends on the phase information of neighboring site (fluid or solid site). This process, which is not included in the LBM algorithm for fluid flow in a simple geometry, brings a heavy memory access to the array which stores phase information, and thus the performance of the algorithm will be seriously damaged. This is one of the problems to be studied in detail in the present paper.

3 Implementation and optimization of multi-GPU based LBM

Similar with the algorithm of Tölke et al. [4], two sets of the distribution functions located in global memory are used. At even time steps, the distribution function values are read from one of them and stored by the other, while at odd time steps the values are shifted in the opposite direction.

To reduce the data access time, the collision and propagation steps are combined in one kernel, named *evolution*(). In this kernel, each thread is assigned to one node, and executed as follows: (1) calculate the index of present node; (2) load distribution function values; (3) compute the macroscopic density and velocity; (4) perform the collision step and store the distribution function values after propagation step.

GPU based LBM generally adopts the structure-of-array (SOA) data layout to meet the requirement of coalescing memory access, i.e., $f_i((x,y,z),t)$ is stored with the index

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

$(i\times nz\times ny\times nx+z\times ny\times nx+y\times nx+x)$. Under this layout, the computational domain is divided along z direction (major direction). For each sub-domain, two ghost layers are employed, which will be used to exchange data. Providing $(nz\times ny\times nx)$ nodes are assigned to one processor, then a grid with a size of $(nz+2)\times ny\times nx$ is allocated, and the ghost layers are $z=0$ and $z=nz+1$.

3.1 Overlapping of the computation and data transfer

To overlap the computation and data transfer, the sub-domain is further divided into two parts, outer layers ($z=1,nz$) and inner layers ($1<z<nz$), then the two parts are assigned to two CUDA streams [1]. A stream is a sequence of commands which are executed in order, and commands within different streams can be executed concurrently. The stream which deals with the outer layers is in charge of data transfer. In this way, data exchange with the neighboring processor can be overlapped with the computation of inner layers, which can be illustrated by Fig. 1.

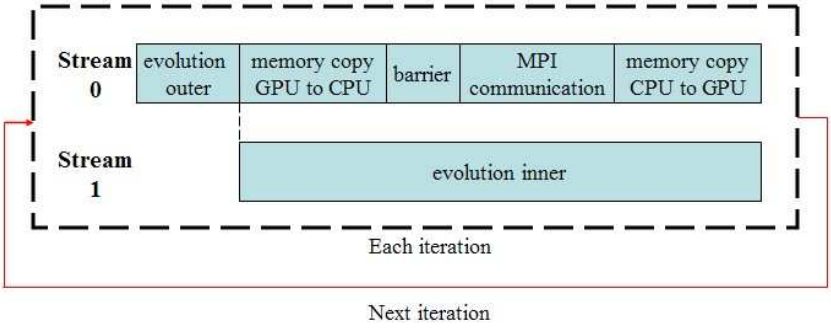


Figure 1: Flow diagram of the overlapped model.

3.2 Access of phase information

To apply the halfway bounce-back scheme for nodes in the computational region, we need to access the phase information of neighboring node in each direction. Generically, an integer is allocated for each node to denote the phase information (fluid or solid site). For D3Q19 model used in the present work, 19 integers need to be loaded while the evolution of each node. To reduce the memory access, the idea proposed by Habich [19] is used. Here we use an integer to store the phase information of all neighboring nodes, the k bit of the integer denotes the phase information of neighboring node in k direction, as shown in Fig. 2.

Another critical issue is the if statements caused by whether to apply the halfway bounce-back scheme or do a normal propagation. These if statements will influence the performance of the algorithm since the GPU is inefficient to handle control flows. To overcome this problem, we firstly calculate the location where to store the distribution

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

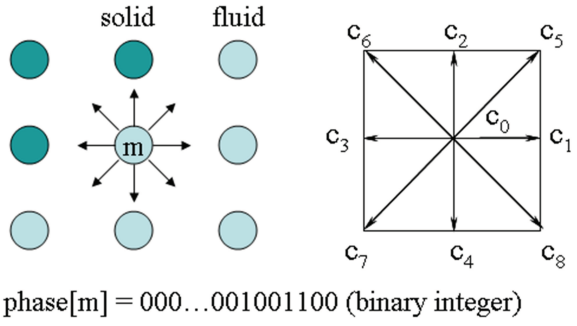


Figure 2: Data structure of the phase information for a two-dimension model.

function values in a uniform manner, and then complete the data storage. For example, for the i direction of node $\mathbf{x}$, the storing location can be given as,

$$add = add(\mathbf{x}+\mathbf{e}_i,i)+((flag \gg i)\&1)\times(add(\mathbf{x},\bar{i})-add(\mathbf{x}+\mathbf{e}_i,i)), \tag{3.1}$$

where $add(\mathbf{x},i)$ is the address of $f_i(\mathbf{x},t+1)$, $\bar{i}$ is the opposite direction of i, $flag$ contains phase information of the neighboring nodes. If the k bit of $flag$ equal to zero, then Eq. (3.1) represents a normal propagation, otherwise, the halfway bounce-back scheme will be applied.

3.3 Communication between processors

The communication between processors can be split into three steps: (1) copy data from global memory to host memory; (2) data exchange through MPI; (3) copy data from host memory to global memory.

After the evolution of the outer layers, the data to be exchanged with neighboring processors are located in ghost layers ($z=0,nz+1$). For each layer, the distribution function values propagating along $+z/-z$ directions need to be copied to host memory and then are exchanged through MPI. After that, the data transferred from other processor are copied back to ghost layers.

It should be noted that we do not copy the exchanged data directly to the outer layer. Although the process of copying the exchanged data to the outer layer seems simpler, it will overwrite the distribution function values which are derived from the halfway bounce-back rule. Due to this issue, some distribution function values of the outer layer are stored at outer layer, but the rest are located at ghost layer. Therefore, special attention needs to be paid to the load of distribution function values in *evolution_outer*() kernel.

For the data transfers between CPU and GPU, five times of data copy are needed for each ghost layer, since the distribution function values of certain z layer in $+z$ or $-z$ directions are not continuous in the memory. Here we adopt a new data layout, where $f_i((x,y,z),t)$ is stored with the index $(z\times 19\times ny\times nx+i\times ny\times nx+y\times nx+x)$. This new data layout maintains the advantage of structure-of-array, and is useful to merge the

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

data copies. Furthermore, the directions of the D3Q19 model are rearranged so that those directions with the same z field are gathered together (see Eq. (3.2)). In this way, the data transfers between CPU and GPU can be completed through one data copy.

$$\mathbf{e}_i = \left\{ \begin{array}{cccccccccc} 0 & 1 & -1 & 0 & 0 & 1 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & \end{array} \right\}. \tag{3.2}$$

4 Performance analysis

In this section, we will test the performance of our algorithm systematically through performing simulations on a one-node cluster equipped with four Tesla C1060 GPU cards using single precision.

4.1 Performance on a single GPU

We first test the performance of our algorithm for Poiseuille flow in a square pipe. According to the test, our algorithm can achieve 446MFLUPS (million fluid lattice updates per second) for simulation of Poiseuille flow (simple geometry) on Tesla C1060 GPU card, which is comparable with the results reported in some published literature [11]. We also run the test on another computer equipped with K20 GPU card and achieve a MFLUPS of 1029 (ECC off). In fact, GPU based LBM is a memory bandwidth limited algorithm. According to the *bandwidthTest* program supplied by the NVIDIA CUDA SDK, the bandwidth of Tesla C1060 and K20 are 72GB/s and 158GB/s, respectively. Thus our algorithm achieve 90% and 94% of peak bandwidth, respectively.

Then simulations of fluid flows through porous media are performed on a single GPU. The porous medium composed of randomly arranged spheres with the same size is used (see Fig. 3). The grid size of the computational domain is 192^3, and porosity of the

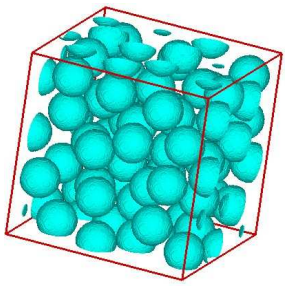


Figure 3: Structure of the porous medium.

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

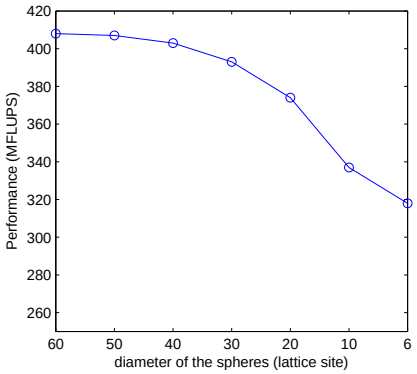


Figure 4: Performance varies with different diameters.

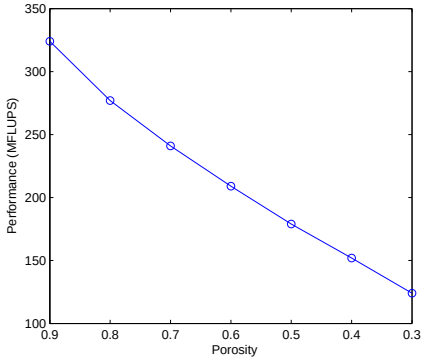


Figure 5: Performance varies with different porosity.

medium is about 80%. We performed several simulations under different diameters of the sphere, and presented the performance in Fig. 4.

As seen from Fig. 4, the performance shows a decline when the sphere diameter getting smaller. This interesting phenomenon can be explained by the following reasons. Firstly, the global memory is accessed by a segment of 32, 64, or 128 bytes, even only one byte in the segment is needed. Thus the distribution function values of solid nodes will be still accessed if fluid node exists in the same segment. The number of the segments, which contain function values of both fluid nodes and solid nodes, will be larger as the diameter getting smaller. As a result, the effective bandwidth will be reduced. Secondly, the execution of halfway bounce-back rule will cause uncoalesced access, which is adverse to performance of the algorithm. While for the cases with smaller sphere diameter, the solid nodes are more evenly distributed and bring more interface nodes, which means the halfway bounce-back rule need to be applied for more nodes.

In addition, we also studied fluid flows through porous media with different porosities. To obtain a large range of the porosity, the diameter of the sphere is set to be 1 lattice (a point). As seen from Fig. 5, the performance shows a significant decline with the de-

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

crease of the porosity. This phenomenon can also be explained by the reasons accounted for the phenomenon shown in Fig. 4.

According to the result of Bernaschi et al. [15], about 170 MFLUPS is achieved using the sparse matrix mode. Comparing with our results, our algorithm based on full matrix mode can achieve a higher performance when the porosity is higher than 50% (see Fig. 5). This result demonstrates that the full matrix mode is a better choice when the medium has a small fraction of obstacles.

From the above tests we can conclude that, the performance of the algorithm will be affected if the solid nodes distribute evenly or if the porosity of the porous medium is low. Our algorithm, which uses full matrix mode, is more appropriate for those porous media with large obstacles and high porosity.

4.2 Performance on multi-GPU platform

Similar to Subsection 4.1, we also tested the performance of our algorithm on multi-GPU platform (four GPU cards) through performing simulations of the Poiseuille flow and fluid flows through porous media. The porous medium with a porosity of 80% is composed of randomly arranged spheres with a diameter of 40 lattices. Two sets of test cases are included. In the first set, a lattice size $192\times192\times48$ is assigned for each GPU; while in the second one, a total lattice size 192^3 is assigned for all the GPUs. The performances of the two test sets are presented in Fig. 6 and Fig. 7, respectively.

From Fig. 6 we can see clearly that our algorithm presents a linear speedup with the number of GPUs, which indicates the excellent scalability of the algorithm. While for the second set (see Fig. 7), the performance is lower than 100% efficiency, which may be caused by the fact that the grid size assigned to each GPU is not large enough to fully exploiting the computing power of the GPU. This can be confirmed by comparing the performance on a single GPU when the grid sizes are $192\times192\times48$ and 192^3, respectively.

At last, we would like to discuss some limitations of our implementation. Firstly, we only employed one-dimensional partition of the computational domain, which is already

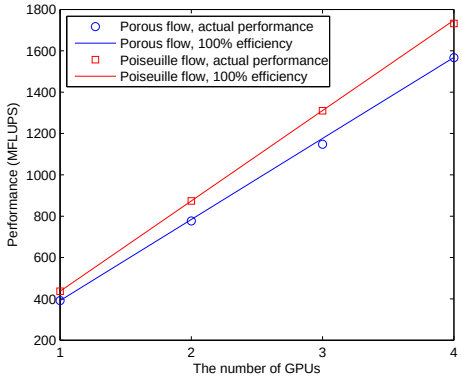


Figure 6: Performance on a lattice size $192\times192\times48$ for each GPU.

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

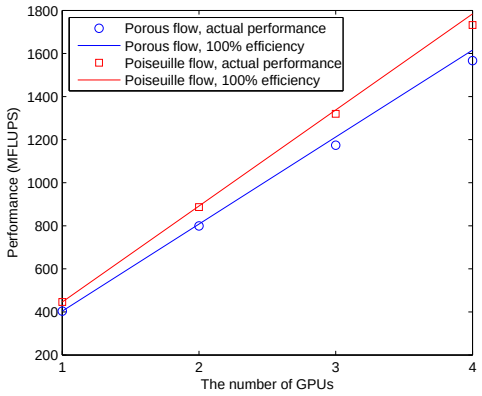


Figure 7: Performance on a total lattice size 192^3.

sufficient for our tested problems. In fact, domain partition along x or y direction (non-major direction) will lead to a large number of data transfer times between GPU and CPU, which damage the performance heavily. This is because the data needed to be exchanged with the sub-domains in these directions are not stored in contiguous memory locations. Even using zero-copy technique provided by CUDA, the payment is still too expensive [9]. However for systems with numerous GPU processors, it is necessary to adopt two- or three-dimensional domain partition to reduce the amount of data to be exchanged. Secondly, our algorithm is only tested on a one-node cluster, and thus further works on the large-scale clusters are still needed, however, the optimization strategies we described in the present paper will still work well.

5 Application

In what follows, as an application of our algorithm, we predict permeabilities of three types of porous medium (see Fig. 8): simple cubic, body-centered cubic, and face-centered cubic arrays of spheres. According to the Darcy law, when the flow velocity is assumed to be sufficiently small, we can derive the permeability as

$$K=-\frac{\mu}{\nabla\bar{P}}\bar{u}, \tag{5.1}$$

where μ is the dynamic viscosity, $\bar{P}$ and $\bar{u}$ are the averaged pressure and velocity, respectively. We carried out several simulations and presented the results in Fig. 9 where the lattice size used is 64^3, the pressure gradient $G=1e-4$, the dimensionless relaxation time $\tau=1.0$, the diameter of the sphere ranges from 12 to 40 to gain different porosities. As shown in this figure, our numerical results agree well with the theoretical solutions proposed by Sangani et al. [20].

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

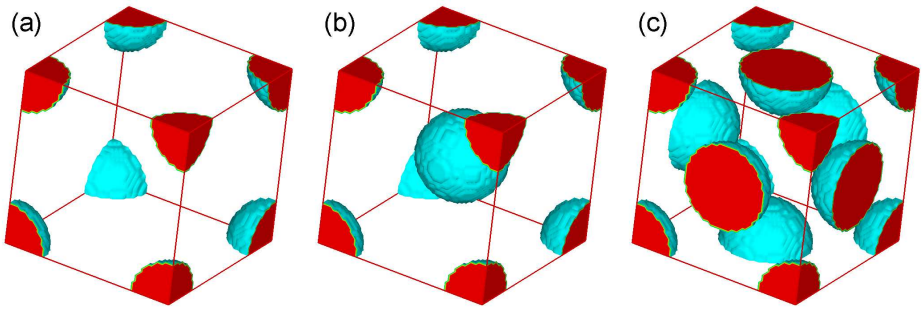


Figure 8: Structure of three cubic arrays of spheres: (a) simple cubic arrays; (b) body-centered cubic arrays; (c) face-centered cubic arrays.

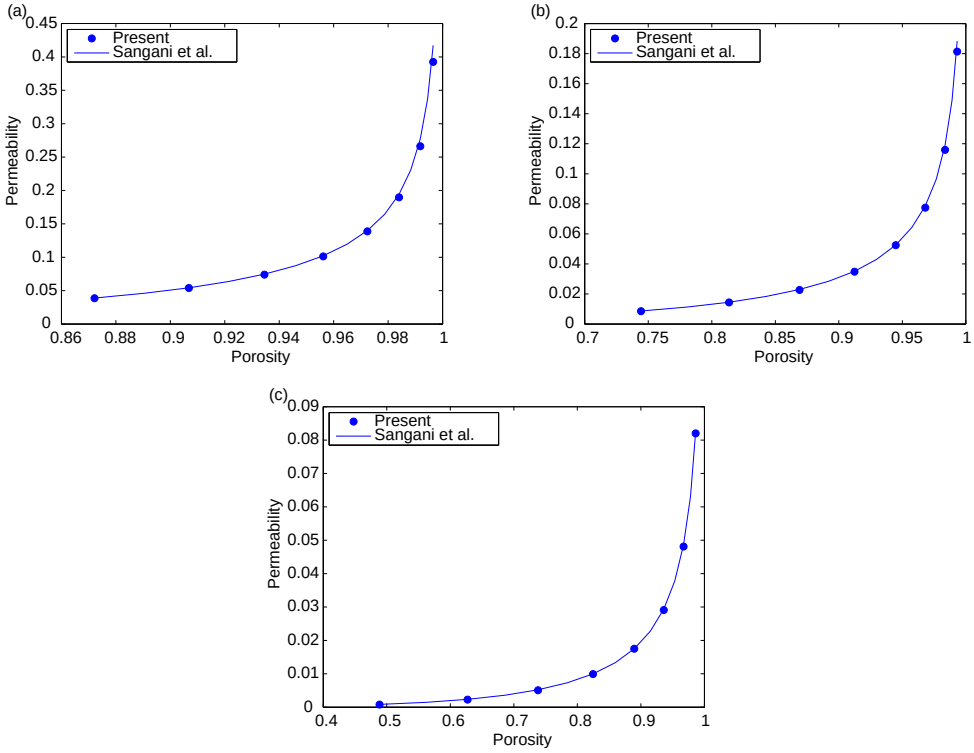


Figure 9: Relation between permeability and porosity: (a) simple cubic arrays; (b) body-centered cubic arrays; (c) face-centered cubic arrays.

6 Conclusions

In this paper, a multi-GPU based lattice Boltzmann algorithm is proposed for fluid flows through porous media by using MPI library to exchange data between processors. Three major optimizations are made in this work: (1) to overlap the computation and data ex-

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

change, asynchronous transfer is realized by using multiple CUDA streams; (2) the data structure used to store phase information is altered to reduce the global memory access using Habich's manner [19]; (3) the data layout is redesigned to merge data copies between GPU and CPU. The numerical results show that our algorithm can achieve a high performance on a one-node cluster equipped with four Tesla C1060 GPU cards, which is up to 1732 MFLUPS for the Poiseuille flow. In addition, a near linear increase with the number of GPUs for both the Poiseuille flow and flow through porous media is also observed. This shows multi-GPU based LBM is a powerful tool to study fluid phenomena in porous media as we shown in Section 5, where we predict the permeabilities of three kinds of porous media and agree well with previous theoretical study. Our results also show that, for porous media with a small obstacle fraction, the full matrix mode can achieve a higher performance than the sparse matrix mode. In the future, we will aim at optimizing the GPU based lattice Boltzmann algorithm on large-scale GPU clusters.

Acknowledgments

This work is financially supported by the National Natural Science Foundation of China (Grant Nos. 11272132, 51006040, and 51006039) and the National Science Fund for Distinguished Young Scholars of China (51125024). Z. C. is also financially supported by the Hong Kong Scholar Program and the China Postdoctoral Science Foundation (Grant No. 2012M521424).

References

[1] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture: Programming Guide (Version 3.2).

[2] R. Benzi, S. Succi and M. Vergassola, *The lattice Boltzmann equation: theory and applications*, Phys. Reports, 222(3) (1992), pp. 145–197.

[3] S. Chen and G. D. Doolen, *Lattice Boltzmann method for fluid flows*, Annual Rev. Fluid Mech., 30(1) (1998), pp. 329–364.

[4] J. Tölke and M. Krafczyk, *TeraFLOP computing on a desktop PC with GPUs for 3D CFD*, Int. J. Comput. Fluid Dyn., 22(7) (2008), pp. 443–456.

[5] F. Kuznik, C. Obrecht, G. Rusaouen and J.-J. Roux, *LBM based flow simulation using GPU computing processor*, Comput. Math. Appl., 59(7) (2010), pp. 2380–2392.

[6] P. Bailey, J. Myre, S. Walsh, D. Lilja and M. Saar, *Accelerating lattice Boltzmann fluid flow simulations using graphics processors*, in: 2009 International Conference on Parallel Processing, IEEE, 2009, pp. 550–557.

[7] J. Habich, T. Zeiser, G. Hager and G. Wellein, *Performance analysis and optimization strategies for a D3Q19 lattice Boltzmann kernel on nVIDIA GPUs using CUDA*, Adv. Eng. Software, 42(5) (2011), pp. 266–272.

[8] C. Obrecht, F. Kuznik, B. Tourancheau and J.-J. Roux, *The TheLMA project: Multi-GPU implementation of the lattice Boltzmann method*, Int. J. High Performance Comput. Appl., 25(3) (2011), pp. 295–303.

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press

[9] C. OBRECHT, F. KUZNIK, B. TOURANCHEAU AND J.-J. ROUX, *Multi-GPU implementation of the lattice Boltzmann method*, Comput. Math. Appl., 65(2) (2013), pp. 252–261.
[10] W. XIAN AND A. TAKAYUKI, *Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster*, Parallel Comput., 37(9) (2011), pp. 521–535.
[11] J. MYRE, S. WALSH, D. LILJA AND M. O. SAAR, *Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters*, Concurrency and Computation: Practice and Experience, 23(4) (2011), pp. 332–350.
[12] Q. G. XIONG, B. LI, J. XU, X. J. FANG, X. W. WANG, L. M. WANG, X. F. HE AND W. GE, *Efficient parallel implementation of the lattice Boltzmann method on large clusters of graphic processing units*, Chinese Science Bulletin, 57(7) (2012), pp. 707–715.
[13] C. OBRECHT, F. KUZNIK, B. TOURANCHEAU AND J.-J. ROUX, *Scalable lattice Boltzmann solvers for CUDA GPU clusters*, Parallel Comput., 39(6-7) (2013), pp. 259–270.
[14] C. FEICHTINGER, J. HABICH, H. KÖSTLER, G. HAGER, U. RUEDE AND G. WELLEIN, *A flexible patch-based lattice Boltzmann parallelization approach for heterogeneous gpu–cpu clusters*, Parallel Comput., 37(9) (2011), pp. 536–549.
[15] 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 and Computation: Practice and Experience, 22(1) (2010), pp. 1–14.
[16] M. BERNASCHI, M. BISSON, M. FATICA, S. MELCHIONNA AND S. SUCCI, *Petaflop hydrokinetic simulations of complex flows on massive GPU clusters*, Comput. Phys. Commun., 184(2) (2013), pp. 329–341.
[17] Y. QIAN, D. D'HUMIERES AND P. LALLEMAND, *Lattice BGK model for Navier-Stokes equation*, Europhys. Lett., 17(6) (1992), pp. 479–484.
[18] X. HE, Q. ZOU, L.-S. LUO AND M. DEMBO, *Analytic solutions of simple flows and analysis of nonslip boundary conditions for the lattice Boltzmann BGK model*, J. Statist. Phys., 87(1) (1997), pp. 115–136.
[19] J. HABICH, Performance Evaluation of Numeric Compute Kernels on nVIDIA GPUs, Master's thesis, University of Erlangen-Nürnberg, 2008.
[20] A. SANGANI AND A. ACRIVOS, *Slow flow through a periodic array of spheres*, Int. J. Multiphase Flow, 8(4) (1982), pp. 343–360.

https://doi.org/10.4208/aamm.2014.m468 Published online by Cambridge University Press