Invited Article Chinese Science Bulletin

Computer Science & Technology

March 2012 Vol.57 No.7: 707–715
doi: 10.1007/s11434-011-4908-y

Efficient parallel implementation of the lattice Boltzmann method on large clusters of graphic processing units

XIONG QinGang[1,2], LI Bo[1,2], XU Ji[1,2], FANG XiaoJian[1,2], WANG XiaoWei[1*], WANG LiMin[1*], HE XianFeng[1] & GE Wei[1]

[1] *State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, Beijing 100190, China;*
[2] *Graduate University of Chinese Academy of Sciences, Beijing 100049, China*

Received May 23, 2011; accepted October 19, 2011

Many-core processors, such as graphic processing units (GPUs), are promising platforms for intrinsic parallel algorithms such as the lattice Boltzmann method (LBM). Although tremendous speedup has been obtained on a single GPU compared with mainstream CPUs, the performance of the LBM for multiple GPUs has not been studied extensively and systematically. In this article, we carry out LBM simulation on a GPU cluster with many nodes, each having multiple Fermi GPUs. Asynchronous execution with CUDA stream functions, OpenMP and non-blocking MPI communication are incorporated to improve efficiency. The algorithm is tested for two-dimensional Couette flow and the results are in good agreement with the analytical solution. For both the one- and two-dimensional decomposition of space, the algorithm performs well as most of the communication time is hidden. Direct numerical simulation of a two-dimensional gas-solid suspension containing more than one million solid particles and one billion gas lattice cells demonstrates the potential of this algorithm in large-scale engineering applications. The algorithm can be directly extended to the three-dimensional decomposition of space and other modeling methods including explicit grid-based methods.

asynchronous execution, compute unified device architecture, graphic processing unit, lattice Boltzmann method, non-blocking message passing interface, OpenMP

Citation: Xiong Q G, Li B, Xu J, et al. Efficient parallel implementation of the lattice Boltzmann method on large clusters of graphic processing units. Chin Sci Bull, 2012, 57: 707–715, doi: 10.1007/s11434-011-4908-y

High-performance computing (HPC) on general-purpose graphical processing units (GPGPUs) has emerged as a competitive approach to make demanding computations such as those of computational fluid dynamics (CFD) [1,2] and discrete particle simulations [3–5]. This is, on one hand, due to the computational capacity of graphical processing units (GPUs), which is almost one order of magnitude higher than that of mainstream central processing units (CPUs) in terms of both peak performance and memory bandwidth, and on the other hand, due to the introduction of effective and convenient programming interfaces such as Compute Unified Device Architecture (CUDA).

*Corresponding authors (email: xwwang@home.ipe.ac.cn; lmwang@home.ipe.ac.cn)

The lattice Boltzmann method (LBM) [6] is a numerical method suitable for GPGPUs owing to its explicit numerical scheme, localized communication mode and inherent additivity of its numerical operations. Hence, it is a powerful alternative to CFD methods such as finite difference and finite volume methods. Implementations of LBM on a single GPU have been reported [7–10] with speedup ratios ranging from tens to above 100 relative to a single CPU core. In the case of multi-GPU implementations, Li et al. [11] performed LBM simulation of lid-driven cavity flow on an HPC system comprising both Nvidia and AMD GPUs, using CUDA and Brook+, respectively, and combining via the Message Passing Interface (MPI). Myre et al. [12] implemented single-phase, multi-phase and multi-component

LBMs on GPU clusters using Open Multi-Processing (OpenMP). In these implementations, data communication between GPUs is trivial or the GPUs are installed at the same node, so the real performances of these implementations were almost unaffected by communication. However, this is not typical in engineering practice. In fact, the data in GPUs cannot be accessed by the network directly and has to be copied, from the GPU to CPU before sending and from the CPU to GPU after receiving, through a PCIe bus with bandwidth of about 10 GB/s currently (Gen 2), which is much lower than that of the GPU global memory. Therefore, communication between the CPU and GPU can be a bottleneck in some applications.

In this article, we integrate asynchronous computing–communication via the CUDA v3.1 framework [13], shared-memory parallelization using OpenMP and inter-node parallelization using non-blocking MPI to improve the performance of multi-GPU LBM simulations. Performances for both one- and two-dimensional decompositions are analyzed and it is found that our implementation is very efficient. The consistency of our implementation on HPC systems with multiple GPUs at one node is emphasized.

1 The lattice Boltzmann method

The lattice BGK model [14] is one of the most frequently used schemes for the LBM. Depending on the dimensionality (D) and number of discrete lattice velocities (Q), there are different variations, such as D2Q9, D3Q13, and D3Q19. The formulation of the lattice BGK model is

$$f_i(\boldsymbol{x}+1,t+1)=f_i(\boldsymbol{x},t)+\frac{1}{\tau}(f_i^{\mathrm{eq}}(\boldsymbol{x},t)-f_i(\boldsymbol{x},t)), \quad (1)$$

where $f_i(\boldsymbol{x},t)$ is the density function of the ith direction at position $\boldsymbol{x}$ and time t. τ is the relaxation time related to fluid molecular dynamic viscosity μ. The term $f_i^{\mathrm{eq}}(\boldsymbol{x},t)$ is approximated to second order as

$$f_i^{eq}(\boldsymbol{x},t)=\rho w_i(1+3\frac{\boldsymbol{e}_i\cdot\boldsymbol{u}}{c}+4.5\frac{(\boldsymbol{e}_i\cdot\boldsymbol{u})^2}{c^2}-1.5\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{c^2}), \quad (2)$$

where

$$\rho=\sum_i f_i,\quad \rho\boldsymbol{u}=\sum_i \boldsymbol{e}_i f_i\ . \quad (3)$$

The D2Q9 scheme is illustrated in Figure 1 and further details were given by Qian et al. [14].

To reduce the compressing effect in the original lattice BGK model, He et al. [15] proposed revisions to the DdQq schemes and named them iDdQq. The evolutional rules are the same but with different equilibrium density propagations:

$$f_i^{eq}(\boldsymbol{x},t)=\rho_0+\lambda_i p$$
$$+w_i(1+3\frac{\boldsymbol{e}_i\cdot\boldsymbol{u}}{c}+4.5\frac{(\boldsymbol{e}_i\cdot\boldsymbol{u})^2}{c^2}-1.5\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{c^2}), \quad (4)$$

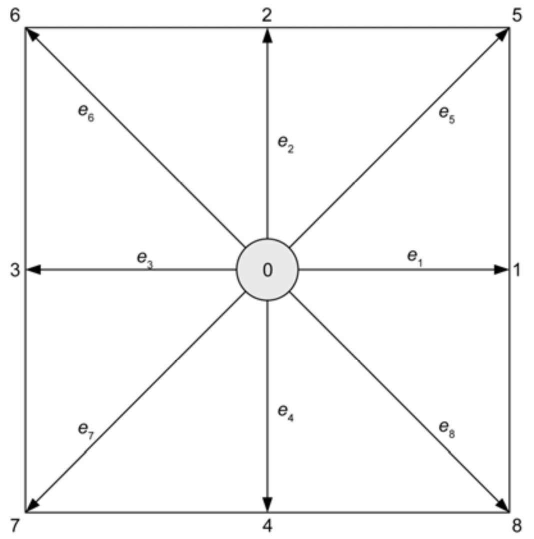

Figure 1 D2Q9 model with w_i = 4/9 when i = 0, w_i = 1/9 when i = 1, 2, 3 and 4, and w_i = 1/36 when i = 5, 6, 7 and 8.

where $\lambda_0=3\frac{w_0-1}{c^2}$, $\lambda_i=3\frac{w_i}{c^2}$, $i\neq 0$. ρ_0 is the referenced fluid density for the initial state, pressure p and velocity $\boldsymbol{u}$ are expressed as

$$p=\frac{c^2}{3(1-w_0)}(\sum_{i\neq 0} f_i-1.5w_0\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{c^2}),\quad \boldsymbol{u}=\sum_{i\neq 0}\boldsymbol{e}_i f_i\ . \quad (5)$$

The iDdQq schemes introduce no further computational cost, and for GPU implementation, the zeroth direction can be omitted, which makes the schemes faster than the corresponding DdQq schemes. However, for iDdQq schemes, the hiding of data communications is more important since the communication-to-computation ratio is higher than DdQq for the size of data to be transfered among GPUs is same.

2 Multi-GPU implementation of the iD2Q9 scheme

The implementation of the LBM for a single GPU has been discussed extensively in [7,16]. We emphasis one point here. As the GPU is suitable for data-independent computation-intensive tasks, the memory access mode is critical to the performance. For this reason, the storage of LBM grid data must be aligned and accessed in a coalescent manner to make full use of the memory bandwidth. As long as global memory access is optimized, the performance of different implementations on the same single GPU varies little. However, for multi-GPU implementation, GPU–CPU data transfer and CPU–CPU communication may require a large portion of the wall time, and they have to be optimized also.

In CUDA 3.1, the launch of a GPU kernel is asynchronous, which means that when a kernel is launched, the system returns to its initial state before the kernel completes its computing. This feature enables the host CPU to perform

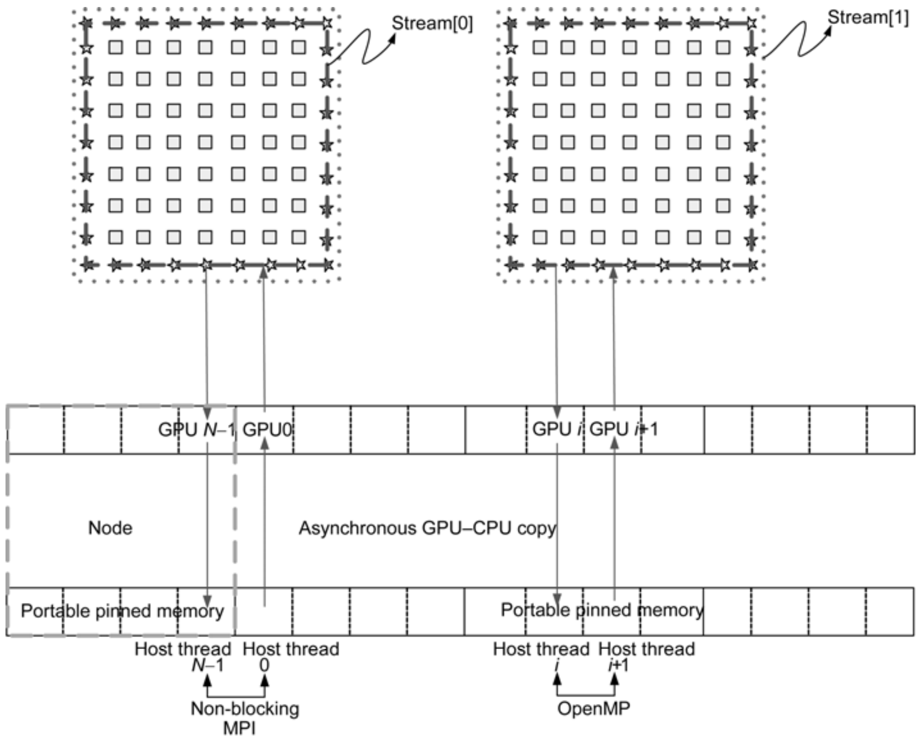

Figure 2 Schematic map of the overlapping of GPU computation and data communications. ☆ indicates a boundary cell and □ an inner cell; ☆ and □ cells make up the entire grid executed in stream [1].

other jobs while waiting for the GPU kernel to finish; e.g., copying data between a GPU and CPU and carrying out inter-CPU communication and arithmetic operations. For LBM simulations, this implies that collision and propagation of the density functions can be run in parallel by copying boundary grid information to a CPU and then transferring the information to neighboring CPUs. As shown in Figure 2, this is realized using the stream function and portable pinned memory in CUDA 3.1, OpenMP and non-blocking communications provided by MPI. The flowchart of parallel implementation of LBM on GPU cluster is given in Figure 3.

At the beginning of each iteration, the collision operation on boundary cells is launched asynchronously by the kernel Boundary_Collision in stream[0]. In this kernel, the boundary grids are only subject to collision and not to propagation, and post-collision boundary information is written to sending buffers in the GPU global memory. The collision and propagation on the entire grid are launched by the kernel Collision_Propagation in stream[1] as soon as Boundary_Collision returns. The host can return before these asynchronous kernels completion, but kernels in the same stream are carried out in series. Therefore, we launch the copy between GPU and CPU cudaMemcpyAsync in stream[0] to ensure that the copy operation starts after the completion of Boundary_Collision. Although the operations in stream[0] are in series, these operations can be done while Collision_Propagation is in execution. To use the asynchronous cudaMemcpyAsync, the buffers in the host must be allocated as pinned memory. After the GPU–CPU copy operation, the communications between CPUs are ready to be carried out. To confirm the finish of GPU–CPU data copy in host memory, cudaStreamSynchronize (stream[0]) is performed to ensure that all boundary information is copied to sending buffers in host memory. Non-blocking MPI_Isend and MPI_Irecv are then launched if the neighboring processors do not belong to the same node. These two MPI functions are non-blocked so that other CPU operations can proceed while data are being sent or received. MPI_Wait is needed to wait until data have been received. If neighboring processors are located on the same node, data can be transfered with the portable pinned memory in CUDA.

This design results in the reduction of the amount of data in MPI and achieves a higher data transfer speed. Such an idea is realized using OpenMP for data communications within a node [17]. OpenMP threads control GPU devices and make portable pinned memory visible to all GPU devices at the same node. Furthermore, a new technology, GPUDirect [18] for Tesla or Fermi GPUs, is adopted to improve communication performance. The improvement is achieved by removing the step of copying data from GPU-dedicated host memory to host memory available to InfiniBand devices to execute the RDMA communications. After the data communications, received data are still copied to the GPU with cudaMemcpyAsync. Finally, the

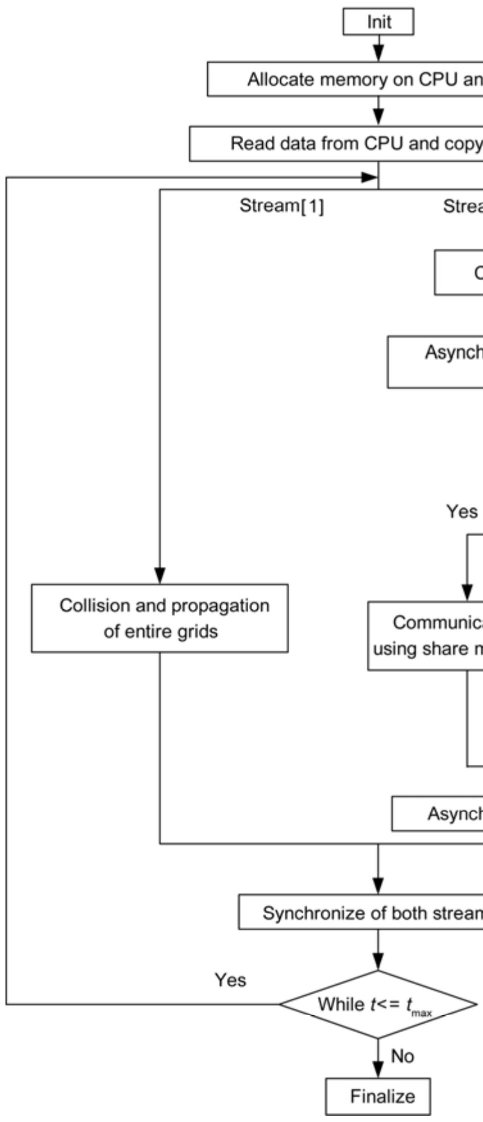
Init
Allocate memory on CPU and GPU
Read data from CPU and copy to GPU
Stream[1]
Stream[0]
Collision of boundary grids
Asynchronous GPU to CPU data copy of boundary grids
Synchronize
Yes
In same node
No
Collision and propagation of entire grids
Communication using share memory
Communication using non-blocking MPI
MPI_Wait()
Asynchronous CPU to GPU data copy
Synchronize of both streams
Yes
While t<= t_{max}
No
Finalize

Figure 3 Flowchart of the hybrid implementation of the LBM on multi-GPUs [20].

boundary information is updated by the data from receiving buffers in GPU global memory.

3 Results and discussion

In the following, the algorithm is validated and its performance tested for our GPU cluster Mole-8.5 (cf. http://www.top500.org/list/2011/11/100), which consists of 362 nodes connected with Quad Data Rate InfiniBand. Most of the computing nodes are equipped with two quad-core CPUs and six Nvidia Tesla C2050 GPUs; therefore, the whole system is configured with more than 2000 GPUs, resulting in peak performance of 2 petaflops in single precision.

3.1 Validation

Numerical validation is important in GPU computing, although many authors [7,19] have declared that the results are insensitive to single precision. We consider the analytical solution for the classical case of two-dimensional Couette flow to evaluate the accuracy of our GPU implementation. The domain size is 2048 × 2048 and the Reynolds number *Re* is 400. The simulation is run in parallel on four GPUs. The simulation results and the analytical solution are illustrated in Figure 4. We find that the computational results of our GPU implementation agree very well with the analytical solution with a maximum error of about 1.5%.

3.2 Performance

Five cases of Couette flow are simulated with the grid sizes for each GPU ranging from 512 × 512 (A), to 512 × 1024 (B), 1024 × 1024 (C), 1024 × 2048 (D) and 2048 × 2048 (E). The whole computation domain is partitioned in either one

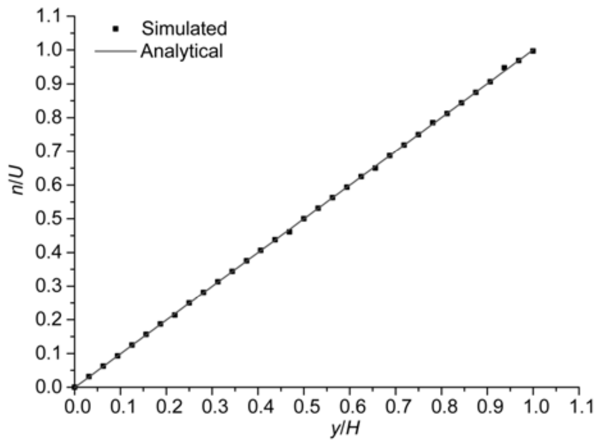

Figure 4 Velocity profiles at steady state for a two-dimensional Couette flow simulation with grid size 2048 × 2048 (Reynolds number $Re = UH/\upsilon = 400$).

or two dimensions. All cases were run 10 times with 10000 iteration steps for each and the wall times were recorded after arithmetical averaging. In the following, unless otherwise specified, each node runs six GPUs concurrently.

Time costs of GPU computation, data transfer between the GPU and CPU and communication between neighboring CPUs in cases using 12 GPUs for one- and two-dimensional decomposition with synchronous execution and blocking MPI are plotted in Figures 5 and 6 respectively. We find that the time portions of GPU–CPU data transfer and communication between CPUs increase with reduction of the domain size for each GPU. In addition, as expected, the time percentage of GPU–CPU and CPU–CPU data transfer in two-dimensional decomposition is higher than that for one-dimensional decomposition and sometimes the time consumption even exceeds the time for GPU computing, which means there is more room to improve the efficiency by hiding data transfer between the GPU and CPU and communications between CPUs.

Simulations deploying the proposed computation–communication overlapping algorithm in both one-and two-dimensional decomposition were carried out. The time costs for all cases are illustrated in Figures 7 and 8. The figures show that most of the time for data copy and communication is successfully hidden through overlapping with GPU computation, leading to an obvious reduction in the total time. In two-dimensional decomposition, the performance improvement is even greater than that in one- dimensional

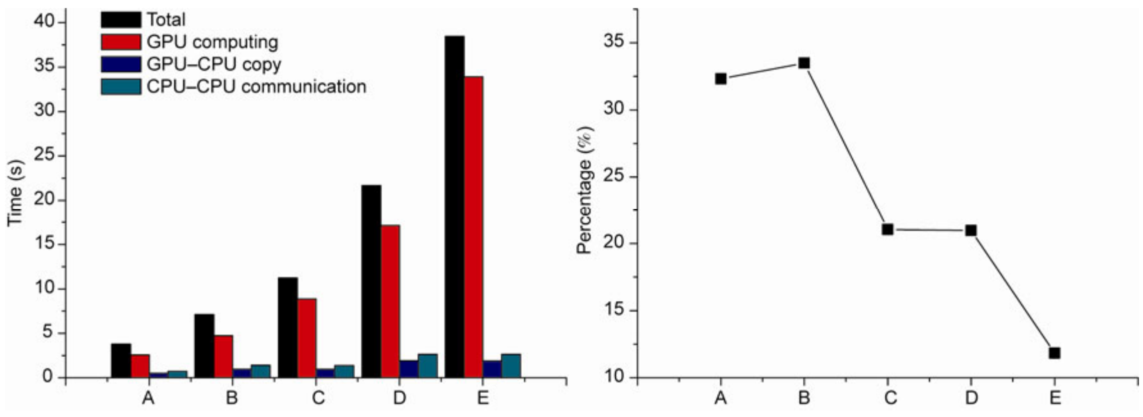

Figure 5 (a) Time component of each part of the algorithm with synchronous execution and blocking MPI but without OpenMP in one-dimensional decomposition; (b) time percentages of GPU–CPU data transfer and CPU–CPU communication.

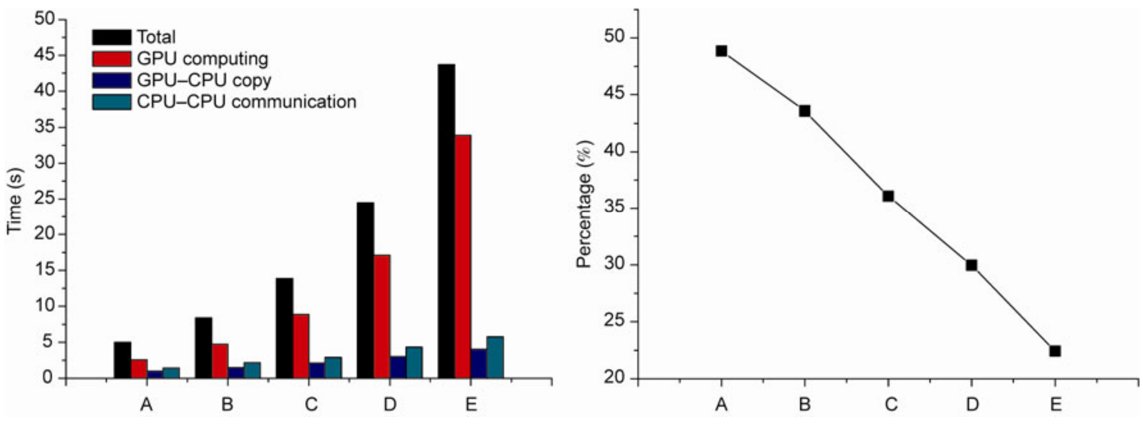

Figure 6 (a) Time component of each part of the algorithm with synchronous execution and blocking MPI in two-dimensional decomposition; (b) time percentages of GPU–CPU data transfer and CPU–CPU communication.

decomposition since more time for data transfer between a GPU and CPU and communication is hidden. To describe the performance improvement clearly, we take case E in one-dimensional decomposition using 12 GPUs as an example to compare time components of 5 algorithms: (a) synchronous execution and blocking MPI without OpenMP; (b) synchronous execution and blocking MPI with OpenMP; (c) asynchronous execution and blocking MPI with OpenMP; (d) synchronous execution and non-blocking MPI with OpenMP; (e) asynchronous execution and non-blocking MPI with OpenMP. The time results are listed in Table 1. Because of the non-serial characteristic of asynchronous execution and non-blocking MPI, the time required for asynchronous GPU execution and non-blocking MPI is difficult to separate. Therefore, the GPU computation time was assumed to be the same for the asynchronous cases. Table 1 shows that the time required for data delivery between the GPU and CPU is reduced by about 60%–70% and the time required for inter-CPU communication is reduced by 70%–80%, which gives performance of 1192 million lattice updates per second for each GPU card in multi-node and multiple GPU implementation.

Table 1 Comparison of time components for five algorithms in case E

Algorithm	GPU computation (s)	GPU–CPU data transfer (s)	CPU–CPU communication (s)	Total (s)
(a)	33.90231	1.89775	2.65466	38.45473
(b)	33.91365	1.88922	1.13562	36.93849
(c)	33.90231	0.63391	1.1479	35.68412
(d)	33.89173	1.90276	0.6431	36.43759
(e)	33.90231	0.63391	0.6431	35.17932

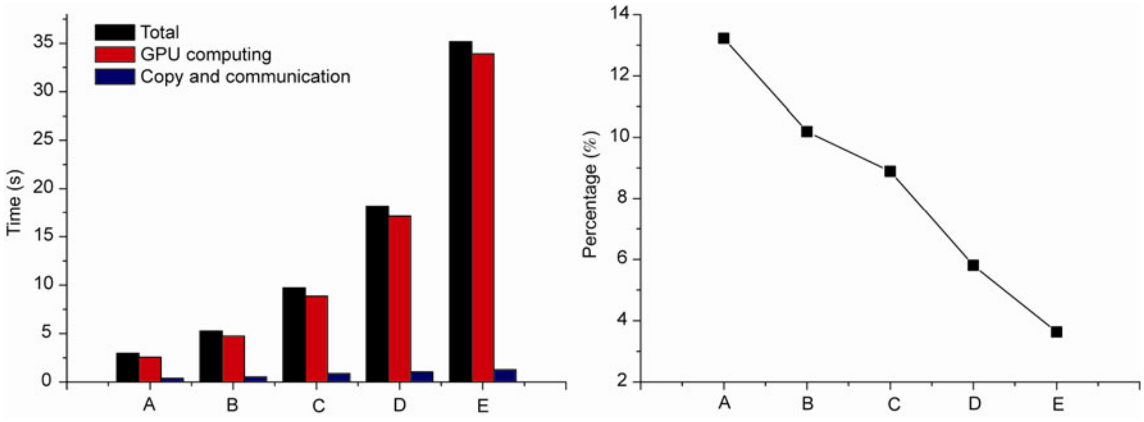

Figure 7 (a) Time component for the algorithm with asynchronous execution, OpenMP and non-blocking MPI in one-dimensional decomposition; (b) time percentage of GPU–CPU copy and CPU–CPU communication.

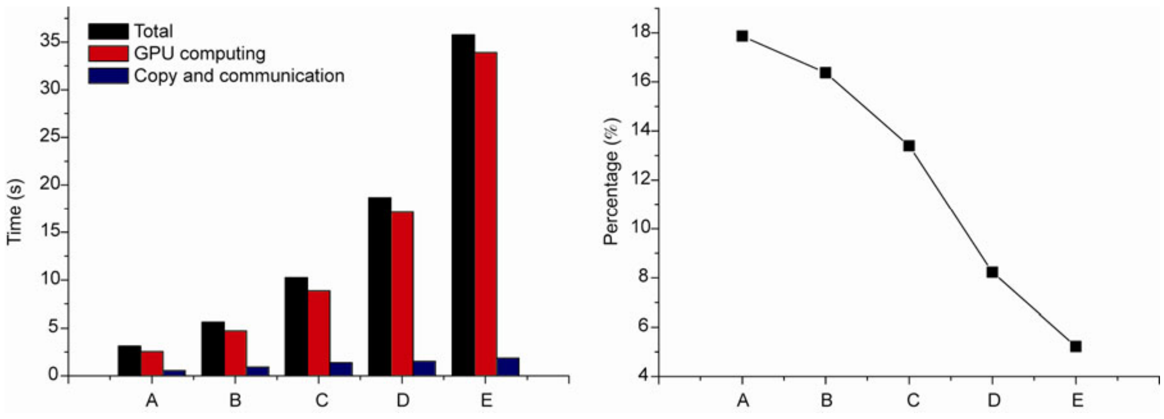

Figure 8 (a) Time component for the algorithm with asynchronous execution, OpenMP and non-blocking MPI in two-dimensional decomposition; (b) time percentage of GPU–CPU copy and CPU–CPU communication.

To investigate the scalability of the implementation further, we change the number of GPUs in case E, ranging from 12 to 1728. The corresponding time costs for communication are shown in Figure 9. We see that the computation–communication overlapping algorithm still performs better than original algorithms with blocking MPI as the number of GPUs increases. This shows that the optimization can be applied to hundreds or thousands of GPUs with good scalability.

3.3 Performance balance for multi-GPUs nodes

In addition to the above performance discussions, we also run our GPU implementation using 12 GPUs for case E but with a varying number (one, two, three, four or six) of GPUs at each node to test the balance of performance and economy for computing nodes integrating multiple GPUs. As it is known that the bandwidth of the PCI-E bus is usually a bottleneck owing to data transfer between the GPU and CPU during computation compared with the GPU computing, the performance deteriorates when multiple GPUs at one node are engaged in a parallel computation because of the PCI-E bandwidth conflict. Owing to the use of CUDA portable pinned memory and OpenMP, the communication load of the processes within a node is theoretically equal, irrespective of how many GPUs are employed concurrently at a node. Therefore, we can ensure that there are negligible differences in the CPU–CPU communication time for the five configuration settings. The performance of our implementation is summarized in Table 2. We find that although the number of GPUs used at each node increases from one to six, the increase in the total computation time is almost negligible as most of the time for communication and data transfer is hidden owing to the asynchronous execution. The time difference is mainly due to the GPU–CPU data transfer as more data are transfered through the PCI-E bus in the case that more GPUs are running on the same

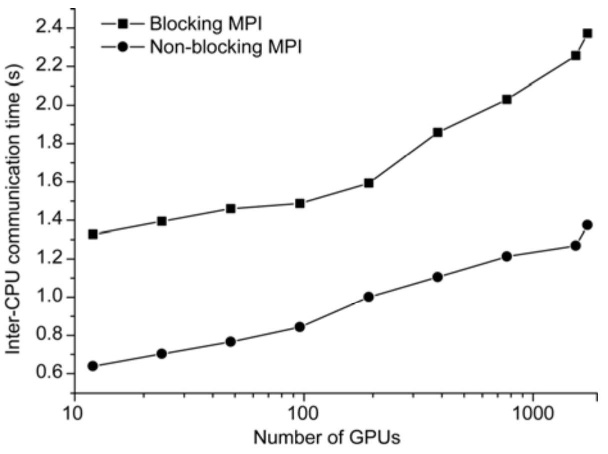

Figure 9 Comparison of communication time between blocking and non-blocking MPI in large-scale LBM simulations.

node. Therefore, we believe that nodes integrating more GPUs like Mole-8.5 achieve a good balance between performance and economy for some applications with an efficient algorithm considering the hardware cost and space occupation.

3.4 Application

Because of CUDA's interoperability with OpenGL, we couple the efficient GPU implementation of the LBM with a visualization framework developed by our group [20] to realize large-scale simulations. In this section, we conduct a direct numerical simulation of gas up-flowing through 1166400 suspended solid particles under a two-dimensional doubly periodical boundary condition. The simulation domain is 11.5 cm× 46 cm, which is discretized by about one billion lattice cells. We simulate the gas-solid flow using 576 GPUs at 96 nodes by two-dimensional domain decomposition. In Figure 10, distinct regions of particle aggregation, which are called clusters in the chemical community, are reproduced. This large-scale simulation confirms that the efficient multi-GPU parallel LBM simulation with a powerful GPU cluster is a promising tool for scientific or industrial modeling.

4 Conclusions and prospects

A hybrid parallel GPU implementation for LBM simulation was proposed. Asynchronous GPU execution technology was applied to confirm overlapping between GPU–CPU data transfer and GPU computation, indicating that a large portion of the time for GPU–CPU copy can be hidden. Data transfer between CPUs is realized with MPI. To hide this inter-CPU communication cost, non-blocking MPI was used to enable concurrent executions of GPU computing and MPI sending and receiving. A shared memory model such as OpenMP was applied to improve the performance of nodes integrated with multiple GPUs. In our test cases, the time required for GPU–CPU data transfer and inter-CPU communication was reduced by up to about 70% for one-dimensional decomposition and 80% for two-dimensional decomposition. These results show that the hybrid multi-GPU LBM implementation is a feasible way to improve efficiency. Large-scale direct numerical simulation of an 11.5 cm× 46 cm two-dimensional doubly periodical gas-solid suspension was demonstrated by coupling the implementation with a visualization framework. The hybrid mode was easy to implement and can be extended to three-dimensional decomposition. Although our implementations were based on the LBM, other CFD methods such as the finite difference and finite volume methods can be incorporated into this hybrid mode and we believe that they will also perform well.

Table 2 Time costs for GPU–CPU data transfer and CPU–CPU communication with a varying number of GPUs at each node in case E

Number of GPUs in a node	GPU computation (s)	GPU–CPU data transfer (s)	CPU–CPU communication (s)	Total (s)
1	33.90231	0.4678	0.6431	35.01321
2	33.90231	0.5307	0.6431	35.07611
3	33.90231	0.5735	0.6431	35.11891
4	33.90231	0.61142	0.6431	35.15683
6	33.90231	0.63391	0.6431	35.17932

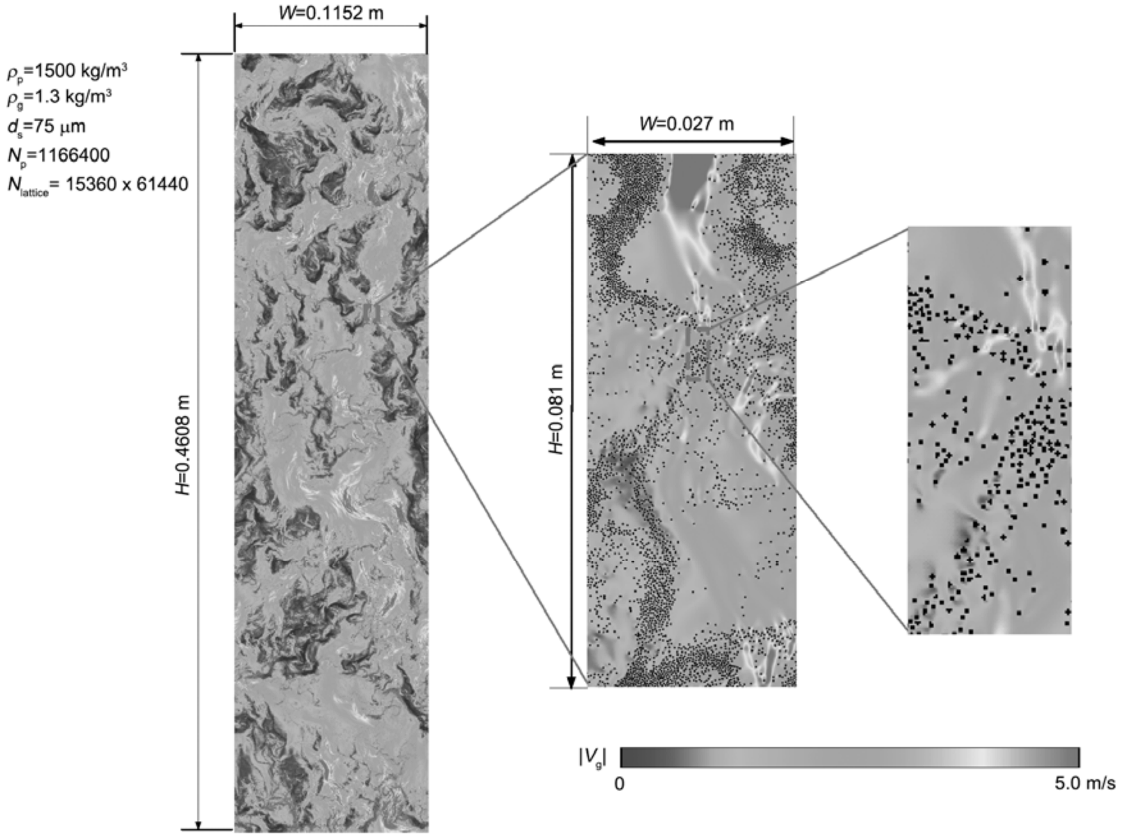

Figure 10 Large-scale direct numerical simulation of a two-dimensional gas-solid suspension containing more than one million particles [20].

This work was supported by the National Natural Science Foundation of China (20221603 and 20906091). We are grateful to Prof. Aibing Yu of University of New South Wales for illuminative discussions. Two anonymous reviewers who gave valuable comments and suggestions that helped improve the quality of this article are gratefully acknowledged. Support from Nvidia through the CUDA Center of Excellence Program is also appreciated.

1 Kampolis I C, Trompoukis X S, Asouti V G, et al. CFD-based analysis and two-level aerodynamic optimization on graphics processing units. Comput Method Appl M, 2010, 199: 712–722
2 Wang J, Xu M, Ge W, et al. GPU accelerated direct numerical simulation with SIMPLE arithmetic for single-phase flow. Chin Sci Bull, 2010, 55: 1979–1986
3 Anderson J A, Lorenz C D, Travesset A. General purpose molecular dynamics simulations fully implemented on graphics processing unit. J Comput Phys, 2008, 227: 5342–5359
4 Chen F, Ge W, Li J. Molecular dynamics simulation of complex multiphase flow on a computer cluster with GPUs. Sci China Ser B: Chem, 2009, 52: 372–380
5 Xiong Q, Li B, Chen F, et al. Direct numerical simulation of sub-grid structures in gas-solid flow—GPU implementation of macro-scale pseudo-particle modeling. Chem Eng Sci, 2010, 65: 5356–5365
6 McNamara G R, Zanetti G. Use of the Boltzmann equation to simulate lattice-gas automata. Phys Rev Lett, 1988, 61: 2332–2335
7 Tolke J, Krafczyk M. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. Int J Comput Fluid D, 2008, 22: 443–456
8 Ge W, Chen F, Meng F, et al. Multi-scale Discrete Simulation Parallel Computing Based on GPU (in Chinese). Beijing: Science Press, 2009
9 Bernaschi M, Fatica M, Melchionna S, et al. A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. Concurr Comp-Pract E, 2010, 22: 1–14
10 Kuznik F, Obrecht C, Rusaouen G, et al. LBM based flow simulation using GPU computing processor. Comput Math Appl, 2010, 59: 2380–2392
11 Li B, Li X, Zhang Y, et al. Lattice Boltzmann simulation on Nvidia

and AMD GPUs (in Chinese). Chin Sci Bull (Chin Ver), 2009, 54: 3177–3184

12 Myre J, Walsh S, Lilja D, et al. Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters. Concurr Comp-Pract E, 2010, 23: 332–350

13 NVIDIA. NVIDIA CUDA compute unified device architecture Programming Guide Version 3.1, 2010

14 Qian Y, Humieres D, Lallemand P. Lattice BGK for Navier-Stokes equation. Europhys Lett, 1992, 17: 479–484

15 He N, Wang N, Shi B. A unified incompressible lattice BGK model and its application to three-dimensional lid-driven cavity flow. Chin Phys, 2004, 13: 40–46

16 Obrecht C, Kuznik F, Tourancheau B, et al. A new approach to the lattice Boltzmann method for graphics processing units. Comput Math Appl, 2011, 61: 3628–3638

17 Yang C, Huang C, Lin C. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. Comput Phys Commun, 2011, 182: 266–269

18 Mellanox. NVIDIA GPUDirect™ Technology——Accelerating GPU-based Systems. 2010

19 Komatitsch D, Erlebacher G, Goddeke D, et al. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. J Comput Phys, 2010, 229: 7692–7714

20 Ge W, Wang W, Yang N, et al. Meso-scale oriented simulation towards virtual process engineering (VPE)—The EMMS paradigm. Chem Eng Sci, 2011, 66: 4426–4458