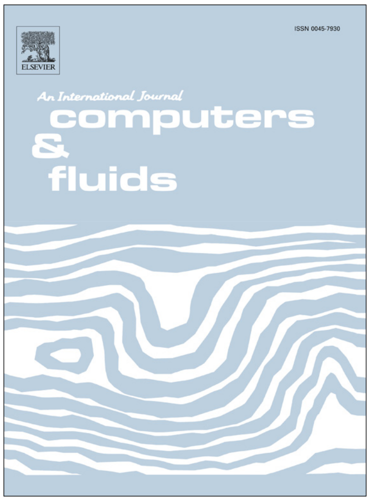
Accepted Manuscript



Parallel computation of entropic lattice boltzmann method on hybrid CPU-GPU accelerated system

Yu Ye, Kenli Li, Yan Wang, Tan Deng

PII: S0045-7930(14)00238-2
DOI: http://dx.doi.org/10.1016/j.compfluid.2014.06.002
Reference: CAF 2581

To appear in: *Computers & Fluids*

Received Date: 7 December 2013
Revised Date: 10 May 2014
Accepted Date: 2 June 2014

Please cite this article as: Ye, Y., Li, K., Wang, Y., Deng, T., Parallel computation of entropic lattice boltzmann method on hybrid CPU-GPU accelerated system, *Computers & Fluids* (2014), doi: http://dx.doi.org/10.1016/j.compfluid.2014.06.002

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Parallel computation of Entropic Lattice Boltzmann method on hybrid CPU-GPU accelerated system

Yu Ye[a], Kenli Li[a,b,*], Yan Wang[a], Tan Deng[a]

[a]College of Information Science and Engineering,Hunan University,Changsha,China
[b]National Supercomputing Center of Changsha,Changsha,China

Abstract

Nowadays, an increasing number of researchers have demonstrated a tendency to choose hybrid CPU-GPU hybrid computing as a high performance computing alternative. Entropic Lattice Boltzmann method(ELBM) parallelization, like many parallel algorithms in the field of rapid scientific and engineering computing, has given rise to much attention for applications of computational fluid dynamics. This study aims to present an efficient implementation of ELBM flow simulation for the D3Q19 model in a hybrid CPU-GPU computing environment, which consists of AMD multi-core CPUs with NVIDIA Graphics Processing Units (GPUs). To overcome the GPU memory size limitation and communication overhead, we propose a set of techniques for the development of an efficient ELBM algorithm for hybrid CPU-GPU computation. Considering the contribution of computational capacity for both the CPU and GPU, an efficient load balancing model is built. The efficiency and accuracy of the proposed approach and established model are

ACCEPTED MANUSCRIPT

*Corresponding author.Tel.:+86 13036781568
Email address: `lkl510@263.net` (Kenli Li)

ACCEPTED MANUSCRIPT

tested on a hybrid CPU-GPU accelerated system, where the intensive parts of the computation are dealt with the software framework OpenMP and CUDA. Finally, we show the comparison of resulting computational performance using a hybrid CPU-GPU approach against both a single CPU core and a single GPU device.

Keywords: Parallel computation, Computational fluid dynamics, Entropic Lattice Boltzmann Method(ELBM), Hybrid computing, CUDA, OpenMP, GPU

---

1. Introduction

Driven by the demands of many complex computational problems with increased accuracy and enhanced numerical performance in the field of scientific computing and engineering simulation, the combined usage of various technologies such as semiconductor technology, manufacturing processes, and power consumption has substantially evolved over the years with remarkable floating point arithmetic performance. The Graphics Processing Unit (GPU) can be considered a computational accelerator possessing a massively multi-threaded architecture which has been widely used for graphical and general purpose computations, such as molecular dynamics simulations and computational fluid dynamics (CFD)[1]. In the early years, General Purpose GPU(GPGPU)[2] computing was performed indirectly by mapping of computational tasks through graphics-related manipulations commonly used by graphics hardware acceleration API's such as OpenGL and DirectX.

In the early years, General Purpose GPU(GPGPU)[2] computing was

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

performed indirectly by mapping of computational tasks through graphics-related manipulations commonly used by graphics hardware acceleration API's such as OpenGL and DirectX. In the present, GPGPU is becoming popular due to its priorities of high, peak performance. Although this programming method was cumbersome, it soon became apparent that the potential computing capability of GPUs could be utilized for accelerating these applications, with the added benefit of having considerably lower cost than supercomputers or workstation clusters at the time. Moreover, the GPU has become increasingly accessible with the release of CUDA (Compute Unified Device Architecture) by NVIDIA. In particular, the advent of the CUDA programming environment[3, 4] has made it possible and convenient to use the computing power of NVIDIA GPU with a simple extension to the standard C language. Despite GPU possessing the above advantages, its real performance is not necessarily higher than that of the current high-performance CPU, especially with recent trends towards multi-core and many-core. The reason for lower real performance of GPU can be illustrated from two aspects. Firstly, the communication overhead of memory transfer between CPU and GPU has a huge impact on raw GPU performance improvements. Secondly, the memory bandwidth limitation is to some extent aggravated due to GPU memory size and its usage restriction limit the maximum data size, which can be computed by the data transfer from CPU to GPU. It is noteworthy that manually conducting proper load balancing under a hybrid CPU-GPU environment for arbitrary data size turns into a hot topic and also remains a difficult challenge.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

The entropic Lattice Boltzmann method (ELBM) originated from the lattice Boltzmann method (LBM), which is a mesoscopic numerical method that simulates macroscopic incompressible fluid dynamics based on discrete kinetic equations (Lattice Boltzmann Equation, LBE)[5]. ELBM takes into account that the stability of the LBM can be guaranteed by satisfying the second law of thermodynamics, that is to say, Boltzmann H-theorem[6–9]. Comparisons of ELBM differ from LBM mainly include that both the local equilibrium distribution function and the relaxation parameter should be restricted by H-theorem per lattice. More specifically, on one hand, the equilibrium state in the ELBM is not explicitly needed because of the collision operators can be computed based on knowledge of H function[10]. In order to ensure the application of H-theorem, one must first find the kinetic state after particles' collision that does not increase entropy during the collision procedure, and then this kinetic state returns a limit for the new state after the collision. On the other hand, ELBM has to be solved the nonlinear equation at each lattice site and every time step[11]. This is extremely computationally expensive. Hence, the development of efficient parallel algorithms and optimized way to solve such a problem has become an important research direction. For LBM parallelization, owing to its simple kernel structure and natural parallelism, LBM has been successfully implemented on multi-core CPU and GPU by many researchers from related fields for high-performance computing in the past few years. As early as in 2003, Lie et al.[12] obtained first promising results of LBM based flow solver on GPU, applying the OpenGL graphics API. The other studies about LBM on a single

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

GPU have been reported[13–17] with good speedup ratios relative to a single CPU core using stencils with different discrete distributions. In the case of LBM implementation for multi-GPU and heterogeneous CPU-GPU clusters, Wang et al.[18] performed simulations of lid-driven cavity flow on a HPC system named Tsubame comprising 170 NVIDIA Tesla S1070 boxes. Christian et al.[19] developed an approach for heterogeneous simulations on clusters equipped with varying node configurations, using WaLBerla framework. In terms of ELBM parallelization and optimization strategies, there are a few research works[20, 21] in order to reduce the computational overhead, most of them are improved from the numerical calculation aspect. Considering the point of view of parallel computing capabilities, the research work of ELBM parallelization is scarce relatively[22].

In the present study, D3Q19-ELBM flow simulation is executed in the hybrid CPU-GPU computing environment. We start with a brief description of ELBM in Section 2. Section 3 gives an overview of CPU-GPU synergetic computing and programming environment via the CUDA v4.2 framework, shared memory parallelization using OpenMP to improve the performance of hybrid CPU-GPU ELBM simulations. Section 4 describes the detailed implementation of D3Q19 flow simulation for combining GPU and multi-core CPU in ELBM hybrid parallelization algorithm, including optimal load balancing model, data structure design and mechanism of task-level parallelism. In Section 5, some comparisons and analyses are made from the performance of the simulations, which are performed on a single CPU core, a singe GPU and hybrid CPU-GPU. Besides, the decomposition mode of computation

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

and communication is presented and compared with the non-decomposition mode. Finally, Section 6 gives a few conclusions and directions for future research.

2. Entropic lattice Boltzmann method and discrete model

Work pertaining to LBM had been applied to a large variety of applications before ELBM was proposed. As we all know, LBM is aimed to model a physical system in terms of the dynamics of fictitious particles (or mass distribution functions) and the macroscopic quantities (such as mass density and momentum density) that can be obtained by evaluating the flow dynamic moments of the distribution function. Generally speaking, LBM can be split into collision and propagation steps, which are required to deal with the density distribution function. However, the non-positivity of distribution function in this method may cause the phenomenon of numerical instablities[23, 24]. To conquer this problem, Karlin et al in some articles[6, 8, 9, 25] put forward an improved scheme that named ELBM that satisfied the second principle of the thermodynamic by imposing the monotonicity and the minimality of H function.

2.1. Main points of ELBM

To develop the entropic LBM, the velocity space and time must be discretized, and then ELBM solves the evolution equation[26, 27] (in lattice units, δt means the time step),

$$f_i(\boldsymbol{x} + \boldsymbol{e_i}\delta t, t + \delta t) = f_i(\boldsymbol{x}, t) + \alpha\beta(f_i^{eq}(\boldsymbol{x}, t) - f_i(\boldsymbol{x}, t)) \quad (1)$$

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

where $f_i(\boldsymbol{x}, t) = f(\boldsymbol{x}, \boldsymbol{u}, t)|_{\boldsymbol{u}=\boldsymbol{e_i}}$, for $i = 0 \dots q-1$ are the density distribution fuctions at lattice site $\boldsymbol{x}$ at time t, corresponding to the q lattice speeds $\boldsymbol{e}_i$. On the left-hand side of Eq.(1) represents the particle free-streaming, whereas the right-hand side includes the particle collisions via the effective relaxation frequency with α as defined by the solution of the following non-linear equation Eq.(2) and β as defined by Eq.(3).

$$H(\boldsymbol{f}) = H(\boldsymbol{f} + \alpha(\boldsymbol{f^{eq}} - \boldsymbol{f})), \boldsymbol{f} = \{f_i\}_{i=0}^{q-1}, \boldsymbol{f^{eq}} = \{f_i^{eq}\}_{i=0}^{q-1} \tag{2}$$

$$\beta = c_s^2 \frac{\delta t}{2\nu + \delta t} \tag{3}$$

where $c_s = \frac{1}{\sqrt{3}}$ is the speed of sound and ν is kinematic viscosity.

In addition, the local equilibrium function f_i^{eq} are derived as the extremum of the following discretized H function[27]:

$$H(\boldsymbol{f}) = \sum_{i=0}^{q-1} f_i \ln(\frac{f_i}{\omega_i}) \tag{4}$$

here the ω_i are the weights associated with each lattice direction in Eq.(4). In the case of isothermal, the explicit expression of the f_i^{eq} is the product of three times the one-dimensional solution and given by

$$f_i^{eq}(\boldsymbol{x}, t) = \rho\omega_i \prod_{d=1}^{D} (2 - \sqrt{1 + 3u_d^2})(\frac{2u_d + \sqrt{1 + 3u_d^2}}{1 - u_d})^{v_{id}} \tag{5}$$

where d is the index of the spatial directions, ρ is fluid density and $\boldsymbol{u} = \{u_d\}_{d=1}^{D}$ are flow velocity, with D physical dimensions and $v_{id} \in \{-1, 0, 1\}$. The above H function is calculated under the constraint of mass and momentum conservation, that is,

$$\rho = \sum_{i=0}^{q-1} f_i = \sum_{i=0}^{q-1} f_i^{eq}, \rho\boldsymbol{u} = \sum_{i=0}^{q-1} \boldsymbol{e_i} f_i = \sum_{i=0}^{q-1} \boldsymbol{e_i} f_i^{eq} \tag{6}$$

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

It is worth mentioning that Eq.(2) can be solved by numerical methods, typically with the Newton-Raphson(NR) method. These operation need to spend a lot of computational cost. Therefore, the present work aims to overcome this difficulty through the rapid development of hardware resources, such as multi-core CPU or GPU.

2.2. D3Q19 discrete model

Indeed, there exists some relevant concept already mentioned in the previous subsection. Among the above described many equation, e.g. Eq.(1), Eq.(2) and Eq.(5), DdQq models for d spatial dimensions and q velocities have been widely adopted by most flow simulations. In the present study, we shall refer to the D3Q19 discretization model, i.e. 19 discrete velocities in three spatial dimensions. The particle velocity $\boldsymbol{e_i}$ may be defined as:

$$\boldsymbol{e_i} = \begin{cases} (0,0,0), & i = 0, \\ (\pm1,0,0)e, (0,\pm1,0)e, (0,0,\pm1)e & i = 1, ..., 6 \\ (\pm1,\pm1,0)e, (\pm1,0,\pm1)e, (0,\pm1,\pm1)e & i = 7, ..., 18 \end{cases} \tag{7}$$

with weights $\omega_0 = \frac{1}{3}$, $\omega_i = \frac{1}{18}(i = 1, ...6)$ and $\omega_i = \frac{1}{36}(i = 7, ...18)$. The fundamental quantity e of equation Eq. (7) is generally written as $\frac{\delta x}{\delta t}$ with $\delta x, \delta t$ representing the lattice spacing and the time increment respectively. Fig. 1 visually shows the D3Q19 model.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

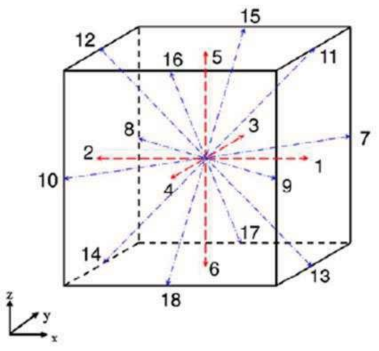


Fig 1: Discrete velocity vectors for the D3Q19 model[18]

3. Overview of CPU-GPU synergetic computing and programming environment

In this section, we will overview of CPU-GPU synergetic computing and programming environment. For hybrid CPU-GPU computing hardware environment, our small high-performance computing workstation, which consists of AMD multi-core CPU with NVIDIA GPU linking through PCI Express X16, often appears in the field of scientific computing. As far as their roles are concerned, GPU is often viewed as the intensive computing core, while CPU collaborate with GPU to transfer data and control program. With respect to software programming environment, most application developers design efficient programs based on OpenMP and CUDA. Specifically, intensive tasks will be allocated and executed on GPU using CUDA, while other tasks adopt other process to control program and migrate data on CPU, besides extra computing tasks.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

3.1. Multi-core CPU programming using OpenMP

OpenMP[28, 29] is an abbreviation for Open Multi-Processing, which is an application programming interface that supports multi-platform shared-memory parallel programming in C/C++ and Fortran, it uses a portable and scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer[30]. OpenMP is designed for multi-processor/core, shared memory machines which may have the underlying architecture including Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA).

OpenMP is composed of several core elements, involving the constructs for thread creation, workload distribution (work sharing), data environment management, thread synchronization, user-level runtime routines and environment variables. For the major components can be described as follows:

- OpenMP provides flexible but simple ways to annotate the program that specific parts can be run in parallel. It uses #*pragma* when C/C++ language is chosen. Usually, the pragma *omp parallel* explicitly instructs the compiler to parallelize the chosen block of code by manual mode. For example, the pragma *omp parallel* is used to fork additional threads to carry out the work enclosed in the construct in parallel. However, the original thread will be denoted as *master thread* with thread ID 0.

- Work-sharing constructs can be used to divide a task among the threads

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

so that each thread executes its allocated part of the code. Under normal circumstances, there are four annotation ways to work independently assigned to one or all of the threads, including *omp for* or *omp do*, *sections*, *single*, *master*.

- Since some private variables are necessary to pass values between the sequential part and the parallel region (i.e. the code block executed in parallel), data environment management and thread synchronization are introduced as data sharing attribute clauses and synchronization clauses by appending them to the OpenMP directive, such as *shared*, *private*, *critical*, *atomic*, etc.

3.2. GPU programming using CUDA

Compute Unified Device Architecture (CUDA) as a general-purpose GPU computing technology developed by NVIDIA has been attracted much attention for more and more scientific and engineering researchers. CUDA capable GPU is composed of a large number of streaming multiprocessors (SM) with multiple streaming processors (SP). Each SP has its own arithmetic units in order to undertake the computational operation. The single-instruction multiple data (SIMD) execution scheme is followed by all streaming processors within one SM, but the overall execution scheme may be described as single-instruction multiple thread (SIMT) due to all SMs are not globally synchronized.

Compared to the CPU memory architecture, CUDA computing devices on GPU show a complex memory hierarchy in Fig. 2 so as to achieve high

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

performance. The total storage of GPU mainly contains a rather large off-chip device memory and a little scarce on-chip memory. An off-chip memory named global memory is shared for all SMs, which is responsible for interaction with the CPU host. When the data is processed by the GPU device, it is first transferred from the host memory to the global memory via calling the command *cudaMemcpy()* with the parameter *cudaMemcpyHostToDevice*. Also, output data from the device required to be transported back to the host memory with the parameter *cudaMemcpyDeviceToHost*. Constant memory also provides interaction with the host, which is not allowed to be written but be read. In addition, texture memory is introduced to accelerate the speed at which frequently used data (stored in texture memory) may be accessed [31][32]. An on-chip shared memory is visible for all the threads executed on a SM and on-chip registers are private to each thread. It has to be remarked that the data communication with on-chip registers and shared memory are much faster than that with off-chip global memory, and shared memories are as fast as registers while also allowing cooperation between threads of the same block. Regrettably, the size of shared memories and register is so small that specific attention is required not to exceed their limited capacities.

Programming in CUDA is easier than traditional GPU, because of it is an extension to C/C++ language. CUDA program can basically be divided into CPU code and one kernel. In fact, the kernel is a void returning function to be executed in several threads with private local variables on the GPU. Threads are grouped together as thread blocks which may have up to three dimensions using the command *dim3 threads()*, they can communicate through the very

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

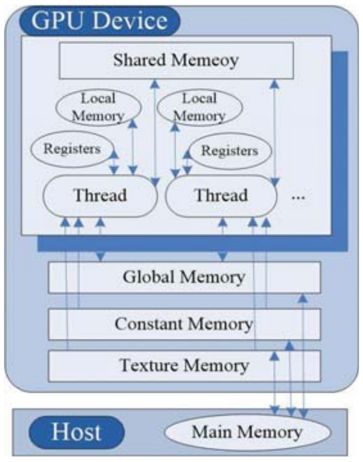


Fig 2: Memory hierarchy in GPU device

fast shared memory in the same block and through the device global memory in different blocks. Blocks are grouped into different dimensional grids using the command *dim3 grid()*, they are executed asynchronously, and there is no efficient dedicated mechanism to ensure global synchronisation. However, global synchronisation is achieved by performing multiple kernel launches.

3.3. CPU-GPU synergetic computing schema

In order to accommodate concurrent execution in the current hybrid CPU-GPU architecture system, we describe herein the synergetic programming schema capable of exploiting the platforms full heterogeneous potentials employing both task and data parallel execution strategies. The general hybrid processing flow of CUDA and OpenMP programming in such a hybrid CPU-GPU environment require the basic steps, as depicted in Fig. 3.

The synergetic programming schema provided herein is constructed with

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

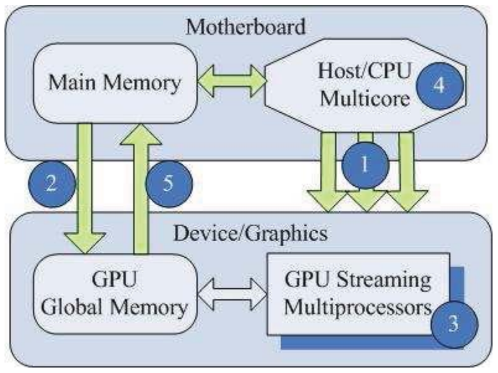


Fig 3: Hybrid programming flow paradigm. ① CPU multi-threaded control GPU. ② CPU Data transfer to Global Memory. ③ GPU parallel computing. ④ CPU parallel computing. ⑤ Result transfer to main memory.

a special focus on providing approaches to overcome the performance challenges of hybrid CPU-GPU accelerated system. First of all, the total task should be divided by theoretical model in order to encourage concurrent execution on hybrid CPU-GPU devices. Secondly, the data structures involved in synergetic computing of ELBM should be different due to different computing environments. Finally, some CUDA device having 2.x computing capability can support device overlapping. They may simultaneously execute a kernel while performing a copy between device and host memory. Achieving these operations requires the use of CUDA streams. The stream in CUDA is a sequence of operations that can execute on the device in the order in which they are issued by the host code. While operations within a stream are guaranteed to execute in the prescribed order, operations in different streams can be interleaved and, when possible, they can even run concurrently[3]. Hence, it is not difficult to achieve the task-level parallelism

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

by using CUDA streams and asynchronous memory copy.

4. Hybrid CPU-GPU implementation of ELBM for D3Q19 model

As mentioned previously in Section 2, the ELBM implementation can be divided into three steps: solving relaxation parameter by H-α solver[22], which needs to satisfy the nonlinear equation Eq.(2). Collision in which the collision operator, i.e. the right side of Eq.(1), is applied to the particle distribution, and propagation in which updated particle populations are propagated to the neighboring nodes as the left side of Eq.(1) based on D3Q19 model. According to our hybrid computing environment, multi-GPU implementation[33] of LBM can be worth learning to achieve. Thus, the ELBM implementation is summarised by the following pseudo-code:

As a matter of fact, the above algorithm can be extended to DdQq models. From the hybrid programming flow paradigm described in Fig 3 and the steps of ELBM implementation for D3Q19 model, the computing process of the individual particles may be both done on the GPU and CPU. For our parallel computation, domain partitioning method based on optimal load balancing allocation model firstly is used in the solution domain. In order to take full advantage of the GPU and CPU computing power respectively, data structure design and memory access model are also concerned with computing environment restrictions. In point of solving the relaxation parameter α, effective methods should be taken. For one thing, the Newton-Raphson iteration method does not work well due to the derivation of the deformation expression in Eq.(2) is too small. For another, there is the phenomenon

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Algorithm 1 ELBM implementation for D3Q19 model

```
1:  for all time step $t$ do
2:      for all lattice node $\boldsymbol{x}$ do
3:          reading velocity distribution $f_i(\boldsymbol{x}, t)$
4:          if node $\boldsymbol{x}$ is on boundaries then
5:              applying boundary conditions to process
6:          end if
7:          computing density $\rho$ and momentum $\rho\boldsymbol{u}$
8:          computing equilibrium distribution $f_i^{eq}(\boldsymbol{x}, t)$
9:          computing relaxation parameter $\alpha$ via $H$-$\alpha$ solver
10:         computing updated distribution $\widetilde{f}_i(\boldsymbol{x}, t)$ by Eq.(1)
11:         computing updated equilibrium distribution $\widetilde{f}_i^{eq}(\boldsymbol{x}, t)$ by Eq.(5)
12:         propagating to neighboring nodes $\boldsymbol{x} + \boldsymbol{e_i}\delta t$
13:     end for
14: end for
```

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

of iteration convergence slowly or no convergence in some grids by using Newton-Raphson iteration method, which impacts on the load imbalance. We hereby use the combination of bisection method and Newton-Raphson iteration method[20] for parallel computing in a hybrid CPU-GPU environment. The usage of a bisection method together with the NR method can solve the relaxation parameter α placidly in the specific search interval[20], even give up seeking the parameter early when the iterations do not converge. Table 1 shows that the combination method is superior to Newton-Raphson iteration method in this case.

Table 1: Comparison of computational time(in seconds) of solving the parameter α with two iteration approaches on single core CPU, single GPU and hybrid CPU-GPU

Mesh size	CPU		GPU		Hybrid CPU-GPU	
	B-N.R.[a]	N.R.[b]	B-N.R.[a]	N.R.[b]	B-N.R.[a]	N.R.[b]
64^3	68.9	75.6	19.2	27.6	15.7	22.8
128^3	562.4	618.8	138.5	177.4	115.2	143.1
256^3	3162.3	3957.2	698.5	752.3	547.9	612.6

[a] B-N.R.: the combination of bisection method and Newton-Raphson iteration method.

[b] N.R.: Newton-Raphson iteration method.

4.1. Model-Based optimal load balancing

To derive the optimal load balancing model, we divide the entire ELBM computation into several sub steps, also including pre-processing stage, i.e.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

memory allocation and data migration. The execution time of these steps are represented with some model parameters, which denote primitive performance of underlying operations and hardware. For a given D3Q19 model of size n^3 lattices and the distribution ratio r to multi-core CPU, we estimate the most appropriate value of r such that the model predicts the shortest execution time.

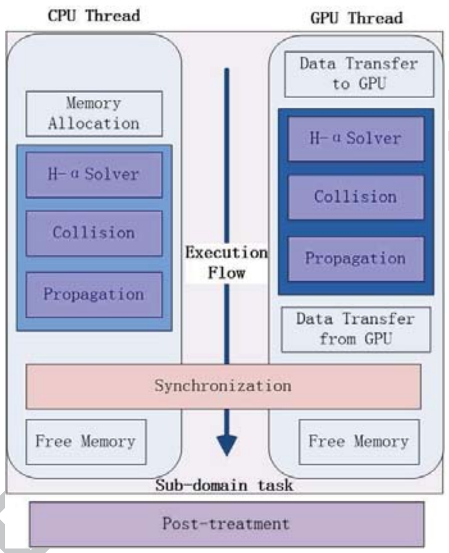


Fig 4: Execution steps in our model of ELBM in hybrid CPU-GPU environment

Fig. 4 illustrates the execution steps of the hybrid CPU-GPU implementation of ELBM for D3Q19 model. We have identified four steps that heavily affect total performance, such as data migration, H-α solver, collision, and propagation. When the lattice size is larger than the available GPU memory, the model may also divide the total domain to sub domains and iterate the steps for each sub domain. Clearly, we first define the optimal load bal-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

ancing model of ELBM in the hybrid system including multi-core CPU and GPU. We define the computational time of the model for the multi-core CPU execution steps as T_{MCPU} using the model parameters, as follows:

$$T_{MCPU} = \frac{r \times n^3 \times (T_{C-COL-PRO} + T_{C-\alpha})}{TD_C} \tag{8}$$

Here, the computational time of propagation in association with collision run on multi-core CPU is marked as $T_{C-COL-PRO}$. Another the crucial sub-step is to solve the relaxation parameter, $T_{C-\alpha}$ and $T_{G-\alpha}$ are represented the H-α solver computational time on multi-core CPU and GPU respectively. In addition, we assume that ELBM simulation is done by using TD_C threads on multi-core CPU in the above model.

Similarly, the GPU computational time model for ELBM is as follows:

$$T_{GPU} = \frac{(1 - r) \times n^3 \times (T_{G-COL-PRO} + T_{G-\alpha})}{TD_G} \tag{9}$$

As equation (8), $T_{G-COL-PRO}$ in equation (9) represents the execution time of particle propagation and collision running on GPU.

Finally, we define the total load balancing model by the sum of time spending on data migration and data computational time. We model the data computational time by the maximum of the multi-core CPU and GPU computing time, since the results require synchronization before post-treatment. Thus, the hybrid CPU-GPU computing model for ELBM is as follows:

$$T = n^3(1 - r)(T_{CTG} + T_{GTC}) + max\{T_{MCPU}, T_{GPU}\} \tag{10}$$

where T_{CTG} denotes the data transmission time from CPU to GPU and the inverse process is denoted as T_{GTC}. It is not difficult to find that the total

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

load balancing model should minimize the total execution time T (i.e Equation 10), when the optimal ratio r can be chosen. In fact, the second term on the right side of Eq.(10) is not so easy to compute due to the difference of computing resources and the size of calculation scale. For the sake of simplicity, we have consciously ignored these factors, which impact on the computational time of propagation in association with collision, and moreover, $T_{C-COL-PRO}$ and $T_{G-COL-PRO}$ are set to the constant values normally. Besides, $T_{C-\alpha}$ and $T_{G-\alpha}$ are generally considered as linear functions of time steps, despite the rapid convergence occurred in some grids.

4.2. Data structure design and memory access model

A key observation that impacts the parallelization and optimization of the applications is that the size of data structures[34]. There are two key aspects for achieving a good ELBM performance in a hybrid CPU-GPU computing environment. One of them is data structure design which can affect the speed of memory access. Array of Structure(AoS)[19] is chosen on multi-core CPU, which means that the particle distribution functions (PDFs) of each cell are stored adjacent in memory. Whereas, for the Structure-of-Arrays(SoA) layout used on GPU, the PDFs pointing in the same lattice direction are adjacent in global memory. As for global memory access, owing to the scattered loads that occur in this way can be efficiently coalesced by the memory sub-system, we can use less shared memory of the GPU. Another problem is the configuration of the execution grid since it may only have up to two dimensions. In our implementation, we choose to use a two-dimensional

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

grid of size $\frac{n^3}{2^q} \times 2^{q-p}$ with one-dimensional blocks of size 2^p, here p and q are free parameters. The optimal values for p and q are determined by the size of lattices.

4.3. Task-level parallelism

In order to hold concurrent execution in the current hybrid environment, we present herein some approaches capable of exploiting the platform's full heterogeneous potentials employing task-level parallelism. Due to different requirements implied by the hybrid CPU-GPU architectural criterion, task as a basic unit of programming should be redefined. As a general rule, a specific application programming is composed by task queues. In particular, the CPU master thread creates a queue of tasks that can be executed according to the algorithm given steps. In the case of the LBM solver, the propagation and collision of particles are performed in parallel by both multi-core CPU and GPU. A queue of tasks is created for executing the procedures. This queue is filled with the appropriate sub-domain grids on the basis of grid size, discretization model and computing resources. Afterwards tasks are performed on multi-core CPU and GPU in an asynchronous manner. Once finishing the corresponding calculation, another task should be pulled from the queue, as is schematically shown in Fig. 5. Thus, both multi-core CPU and GPU are constantly busy with computation until the queue is empty.

For the case of the ELBM implementation for D3Q19 model, the relatively time-consuming operation is the relaxation parameter solution of Eq.(2), i.e. H-α solver [22]. This operation is required for the computation of the precon-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

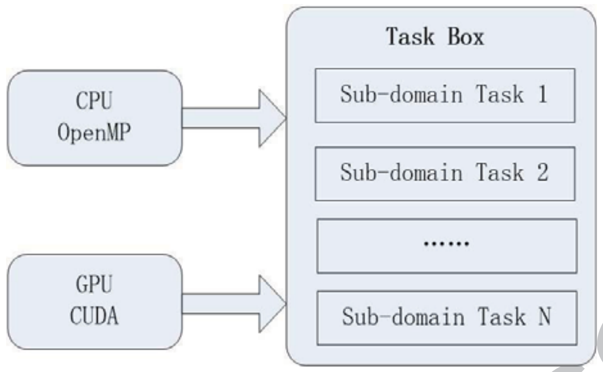


Fig 5: Task queue contains computations to be performed by heterogeneous resources

ditioning steps of the ELBM algorithm, while the propagation and collision are performed after the solution of the relaxation parameter step with H-α solver. For each synchronous CPU threads, they perform the same for-loop body, which consists of the actual function calls. The task that fetched from the queue of tasks can describe the program calculation block, which need to be performed by the function calls. While GPU threads are asynchronous which means that GPU kernels will be launched concurrently, since there is a series of program statements that launches GPU kernels. To allow concurrent data copying and kernel execution of the ELBM implementation for D3Q19 model, our implementations need to use multiple queues, called streams in CUDA. So the tasks in different streams can go in parallel. And moreover, we find that the sizes of sub-domain grid depend on the computing resources of multi-core CPU and GPU constituted, the selection of space-time discretization model and the total grid sizes of simulation scene. The more we use computing cores and the grid sizes, the finer we will partition the sub-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

domain grid. For the aspect of space-time discretization model, the number of dimensions and the number of discrete velocity directions will indirectly affect the division of the sub-domain grid. Eventually, the computational time will shorten as the application of the finer sub-domain grid.

5. Experimental evaluation

We demonstrate the hybrid CPU-GPU synergetic ELBM solver with the 3D lid driven cavity, which has been extensively studied and usually used as a benchmark solution to test the fluid flow. In Section 5.1, we introduce the experiment environment including basic information on the hardware structure and the relevant parameters of the simulation. Subsequently, we investigate the computational time of ELBM on a single core CPU, single GPU and hybrid CPU-GPU computing framework in Section 5.2. Additionally, the Million Lattice Updates Per Second(MLUPS) as another performance index is also discussed. Moreover, the effects of the same lattice size on the accuracy of the numerical solution given by different ELBM implementations are investigated in Section 5.3.

5.1. Experimental environment

To evaluate the effectiveness of our proposed method, two experimental studies are employed to compare the entire ELBM computational time, that is to say, we show that the comparison of performance results using both CPU and GPU with using either a CPU core or a GPU. For both of the studies, we use a two AMD Opteron(TM) processor(2.2GHz), eight-core machine with

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

8GB main memory, and a Tesla C2050 GPU. The GPU is equipped with 448 stream processors running at 1.15GHz processor clock with 3GB of graphics memory. We use Linux kernel 2.6.38 with NVIDIA display driver version 256.53 and CUDA 4.2 and GCC version 4.4.5.

Our numerical simulations are performed for Reynolds number 1000 with different mesh grid sizes. Thereamong, the Reynolds number is defined as: $Re = \frac{LU_{max}}{\nu}$, here ν is the viscosity, L is the cavity side. Only the U_{max} is changed to get fine grids. With the fluid viscosity $\nu = 0.0256$ and $Re = 1000$, $n = 64, 128$ and 256 lattices per side need to be imposed velocity $U_{max} = 0.4, 0.2$ and 0.1 respectively.

5.2. Performance of ELBM with Hybrid CPU-GPU computing

Table 2 compares the total computational time and MLUPS of each implementation for different experimental platforms, scaling the simulations to various domain grid sizes varying from 64^3 to 256^3 cells. The performance of ELBM achieved with hybrid CPU-GPU results between 4 and 6 times faster than ELBM-CPU, depending on the domain size and data distribution ratio. For single core CPU, we obtain the results over 1000 iterations. Nevertheless, the GPU and hybrid CPU-GPU results have been performed for 10000 iterations.

To see how optimal choice of r affects the performance, we plot the speedup of the hybrid CPU-GPU implementation as a function of data distribution ratio r. Here, we fix the mesh grid size n^3 to 128^3 and r varies from 1/512 to 6/512. The optimal data distribution ratio of ELBM solver

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Table 2: Comparison of execution time(in seconds) and MULPS of ELBM implementation on single core CPU, single GPU and hybrid CPU-GPU

Mesh size	CPU		GPU			Hybrid CPU-GPU		
	Total	MLUPS	Total	MLUPS	Speedup	Total	MLUPS	Speedup
64^3	219.2	1.20	58.6	44.73	3.74	49.1	53.39	4.46
128^3	1652.5	1.27	426.5	49.17	3.87	321.3	65.27	5.14
256^3	9586.6	1.75	2136.8	78.52	4.49	1682.5	99.72	5.70

in hybrid CPU-GPU environment for a given grid size, as shown in Fig. 6, is 4/512 in this case. From the trend of line chart in Fig. 6, we can also see that the speedup of hybrid CPU-GPU ELBM solver decrease with increasing of the ratio, and it is because of this that the more data is assigned to CPU for processing, the higher computational time and cost are due to communication overhead and computing capability.

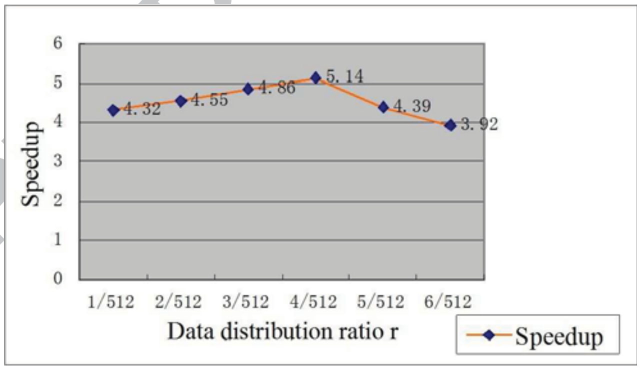


Fig 6: Hybrid CPU-GPU computing speedup as a function of data distribution ratio $r(n^3 = 128^3)$

On the other hand, we further gain insights into the performance factor

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

of data distribution ratio by performing the hybrid CPU-GPU ELBM solver on various grid sizes. Fig. 7 displays the evolution of the communication overhead proportion at different data distribution ratios with increasing grids. This diagram indicates that the choice of data distribution ratio depends on the grid sizes, and can not be generalized. Also, the communication overhead will impact on the overall performance of ELBM with hybrid CPU-GPU computing.

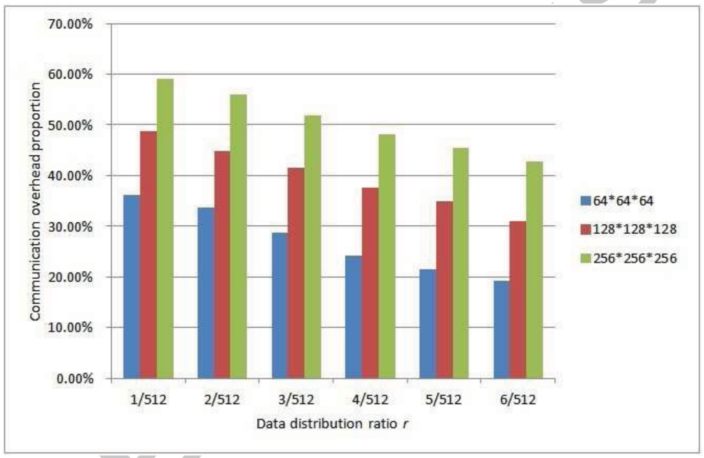


Fig 7: Communication overhead proportion of hybrid CPU-GPU computing at different data distribution ratios

5.3. Numerical error analysis

To evaluate the impact of the same lattice size on the accuracy of the numerical solution, we experiment the simulation of the cavity flow with three implementations of ELBM: CPU-only, GPU-only and hybrid CPU-GPU. As in the previous cases, some physical quantities are fixed in lattice

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

units with $128 \times 128 \times 128$ grids. On the basis of the relative velocity errors put forward in [20], the effects of three implementations are also measured by the error values E_1 and E_2[22], i.e. they are given by:

$$E_1 = \frac{\sum_{x,y}[|u_1(x,y) - u_0(x,y)| + |v_1(x,y) - v_0(x,y)|]}{\sum_{x,y}[|u_0(x,y)| + |v_0(x,y)|]} \tag{11}$$

$$E_2 = \frac{\sqrt{\sum_{x,y}\{[u_1(x,y) - u_0(x,y)]^2 + [v_1(x,y) - v_0(x,y)]^2\}}}{\sqrt{\sum_{x,y}\{[u_0(x,y)]^2 + [v_0(x,y)]^2\}}} \tag{12}$$

where u and v are the x and y components of the velocity and the subscripts 0 and 1 indicate the standard LBM and the test case, respectively. The error values E_1 and E_2 of ELBM simulation for D3Q19 in three ways are little discrepant, as demonstrated in Table 3. The results show that, the hybrid CPU-GPU computing errors are in the middle position amongst three implementations of ELBM, while CPU computing errors are the smallest ones due to single-precision floating-point arithmetic of CPU has more advantages. However, compared with the computing speed of GPU and hybrid CPU-GPU, the drawback of simple CPU is obvious.

Table 3: Values of the errors E1 and E2 computed using three ways(128^3 grids)

Error type	CPU-only	GPU-only	Hybrid CPU-GPU
E_1	0.0685	0.2188	0.1532
E_2	0.1892	0.3296	0.2178

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

6. Conclusions and future work

In this contribution, we have presented the hybrid CPU-GPU computing environment of ELBM, based on mainstream parallel programming platform, i.e. CUDA and OpenMP. With the help of optimal load balancing model, the optimal data distribution ratios among CPU and GPU is chosen. Moreover, this approach not only takes advantage of GPU processing ability for intensive data and parallel data, but also exploits the computing power of CPU adequately. We have tested our hybrid CPU-GPU implementation with 3D lid driven cavity using two AMD Opteron(TM) processor (2.2GHz), eight-core machine with 8GB main memory and a Tesla C2050 GPU. The overall computing time improvements with the optimal load balancing model range from 16.2% to 82.4% compared to the CPU-only and GPU-only configurations.

Our future work includes the following. Firstly, we plan to research many GPUs and multi-core CPU both in hybrid architecture system of ELBM. Specifically, the performance of ELBM implementation with MPI parallelization on large heterogeneous cluster will be given much attention. Secondly, the accuracy of optimal load balancing model needs to further discussion and demonstration in detail. Finally, we will implement the simulation by multi-block entropic Lattice Boltzmann method based on GPU as well as hybrid CPU-GPU computing environment.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Acknowledgements

This work was partially funded by the Key Program of National Natural Science Foundation of China (Grant No. 61133005), the National Natural Science Foundation of China (Grant Nos. 61070057,61103047,61370095) and the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011). An earlier version of this paper was presented at the International Conference on Parallel Computational Fluid Dynamics(ParCFD) Conference 2013. We also thank the anonymous reviewers for their helpful feedbacks.

References

[1] Kampolis I, Trompoukis X, Asouti V, Giannakoglou K. Cfd-based analysis and two-level aerodynamic optimization on graphics processing units. Computer Methods in Applied Mechanics and Engineering 2010;199(9):712–22.

[2] GPGPU.org . General-Purpose Computation on Graphics Hardware. `http://gpgpu.org/`; 2002.

[3] Corporation N. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. `http://developer.nvidia.com/object/cuda.html`; 2013.

[4] Corporation N. NVIDIA Cuda Toolkit 2.3. `http://www.nvidia.com/object/cuda_get.html`; 2009.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[5] Chen S, Doolen GD. Lattice boltzmann method for fluid flows. Annual review of fluid mechanics 1998;30(1):329–64.

[6] Ansumali S, Karlin IV. Stabilization of the lattice boltzmann method by the h theorem: A numerical test. Physical Review E 2000;62(6):7999–8003.

[7] Boghosian BM, Yepez J, Coveney PV, Wager A. Entropic lattice boltzmann methods. Proceedings of the Royal Society of London Series A: Mathematical, Physical and Engineering Sciences 2001;457(2007):717–66.

[8] Ansumali S, Karlin IV. Entropy function approach to the lattice boltzmann method. Journal of statistical physics 2002;107(1-2):291–308.

[9] Karlin IV, Gorban AN, Succi S, Boffi V. Maximum entropy principle for lattice kinetic equations. Physical review letters 1998;81(1):6–9.

[10] Ansumali S, Karlin IV. Single relaxation time model for entropic lattice boltzmann methods. PHYSICAL REVIEW-SERIES E- 2002;65(5; PART 1):056312–.

[11] Yong WA, Luo LS. Nonexistence of h theorems for the athermal lattice boltzmann models with polynomial equilibria. PHYSICAL REVIEW-SERIES E- 2003;67(5; PART 1):051105–.

[12] Li W, Wei X, Kaufman A. Implementing lattice boltzmann computation on graphics hardware. The Visual Computer 2003;19(7-8):444–56.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[13] Tanno I, Hashimoto T, Yasuda T, Tanaka Y, Morinishi K, Satofuka N. Simulation of turbulent flow by lattice boltzmann method and conventional method on a gpu. "Computers & Fluids" 2012;.

[14] Tölke J, Krafczyk M. Teraflop computing on a desktop pc with gpus for 3d cfd. International Journal of Computational Fluid Dynamics 2008;22(7):443–56.

[15] Bernaschi M, Fatica M, Melchionna S, Succi S, Kaxiras E. A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. Concurrency and Computation: Practice and Experience 2010;22(1):1–14.

[16] Kuznik F, Obrecht C, Rusaouen G, Roux JJ. Lbm based flow simulation using gpu computing processor. "Computers & Mathematics with Applications" 2010;59(7):2380–92.

[17] Tölke J. Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by nvidia. Computing and Visualization in Science 2010;13(1):29–39.

[18] Xian W, Takayuki A. Multi-gpu performance of incompressible flow computation by lattice boltzmann method on gpu cluster. Parallel Computing 2011;37(9):521–35.

[19] Feichtinger C, Habich J, Köstler H, Hager G, Rüde U, Wellein G. A flexible patch-based lattice boltzmann parallelization approach for heterogeneous gpu–cpu clusters. Parallel Computing 2011;37(9):536–49.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[20] Tosi F, Ubertini S, Succi S, Karlin I. Optimization strategies for the entropic lattice boltzmann method. Journal of Scientific Computing 2007;30(3):369–87.

[21] Yasuda T, Satofuka N. An improved entropic lattice boltzmann model for parallel computation. Computers & Fluids 2011;45(1):187–90.

[22] Ye Y, Li K. Entropic lattice boltzmann method based high reynolds number flow simulation using cuda on gpu. Computers & Fluids 2013;.

[23] Higuera F, Succi S, Benzi R. Lattice gas dynamics with enhanced collisions. EPL(Europhysics Letters) 1989;9(4):345–60.

[24] Succi S. The lattice Boltzmann equation: for fluid dynamics and beyond. Oxford university press; 2001.

[25] Karlin IV, Succi S. Equilibria for discrete kinetic equations. Physical Review E 1998;58:4053–65.

[26] Ansumali S, Karlin I, Öttinger H. Minimal entropic kinetic models for hydrodynamics. EPL (Europhysics Letters) 2003;63(6):798–810.

[27] Malaspinas O, Deville M, Chopard B. Towards a physical interpretation of the entropic lattice boltzmann method. Physical Review E 2008;78(6):066705–2.

[28] Menon R, Dagum L, Kohr D, Maydan D, McDonald J. Parallel Programming in OpenMP. Morgan Kaufmann; 2000.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[29] OpenMP.org . OpenMP. `http://openmp.org/`; 1997.

[30] OpenMP.org . OpenMP 3.0 Status. `http://openmp.org/wp/2008/11/openmp-30-status/`; 2008.

[31] Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WmW. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. ACM; 2008, p. 73–82.

[32] Farber R. Cuda, supercomputing for the masses: Part. 2008.

[33] Obrecht C, Kuznik F, Tourancheau B, Roux JJ. Multi-gpu implementation of the lattice boltzmann method. Computers & Mathematics with Applications 2013;65(2):252–61.

[34] Papadrakakis M, Stavroulakis G, Karatarakis A. A new era in scientific computing: domain decomposition methods in hybrid cpu–gpu architectures. Computer Methods in Applied Mechanics and Engineering 2011;200(13):1490–508.

ACCEPTED MANUSCRIPT

Highlights

- A very efficient parallel strategy of ELBM on hybrid CPU-GPU system is presented.
- We establish the optimal load balancing model which can improve performance.
- For the given lattices, the best data distribution ratio is proposed.
- We analyze and compare the performance index of ELBM on hybrid computing platform.
- Communication overhead and error analysis of hybrid CPU-GPU system are discussed.