Computers & Fluids 115 (2015) 192–200

Contents lists available at ScienceDirect

Computers & Fluids

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

GPU acceleration of Volumetric Lattice Boltzmann Method for patient-specific computational hemodynamics

Zhiqiang Wang [a], Ye Zhao [a,*], Alan P. Sawchuck [b], Michael C. Dalsing [b], Huidan (Whitney) Yu [c,b,*]

[a] Department of Computer Science, Kent State University, OH 44240, USA
[b] Surgery Division of Vascular Surgery, School of Medicine, Indiana University, IN 46202, USA
[c] Mechanical Engineering Department, Indiana University-Purdue University Indianapolis, IN 46202, USA

ARTICLE INFO

Article history:
Received 3 October 2014
Received in revised form 23 March 2015
Accepted 2 April 2015
Available online 7 April 2015

Keywords:
Volumetric Lattice Boltzmann Method
Parallel computing
Graphics processor unit
Patient-specific computational hemodynamics

ABSTRACT

Volumetric Lattice Boltzmann Method (VLBM) has been recently developed for solving complex flow with arbitrary curved boundaries. The VLBM regards fluid particles are uniformly distributed in cells and distinguishes fluid, solid, and boundary cells by introducing a volumetric parameter $P(\boldsymbol{x}, t)$ defining the percentage of solid volume in each cell. The advantages of VLBM stem from the self-regulation of $P(\boldsymbol{x}, t)$ in the volumetric lattice Boltzmann equation (VLBE) for particle collision and streaming with no spatial interpolation when dealing with an arbitrarily curved boundary with or without motion. First, the VLBE satisfies mass conservation strictly. Second, the implementation of VLBM is rather simple after the solid volume percentages are determined in boundary cells. And third, no-slip boundary condition is integrated in the streaming formulation thus significantly enhances the capability of parallelization. In this paper, we perform GPU (Graphics Processing Unit) parallelization for VLBM using a uniform computing scheme for both fluid and boundary cells. In contrast to the traditional LBM acceleration, the boundary conditions have to be imposed over boundary nodes, where branching operations are required to identify boundary nodes from others, the VLBM implementation does not need to distinguish fluid and boundary cells in the computation so that branching is minimized and the GPU kernel execution is accelerated. Furthermore, the algorithmic steps are optimized to improve coalesced access of GPU memory and avoid race condition. An application study is on a pulsatile blood flow in a patient-specific carotid artery segmented from an anonymous clinical CT image and more than 30 times speedup over the serial counterpart. Simulations of fluid dynamics and wall shear stress (WSS) are presented and known velocity skewness and WSS distributions are captured. The GPU accelerated VLBM is promising to perform patient-specific computational hemodynamics within clinical accepted time frame and is expected to reveal quantitative real-time blood flow in living human arteries to aid clinical assessment of cardiovascular diseases.

1. Introduction

Lattice Boltzmann method (LBM) [1–3] has emerged as a popular alternative computational methodology to traditional Navier–Stokes solvers for a large variety of incompressible and time-dependent complex flows [4,5]. Its strength lies in the ability to easily represent complex physical phenomena, ranging from multiphase flows, porous media, to chemical interactions between the fluid and the surroundings. The LBM deals with a discrete mesoscopic equation governing the time evolution of discrete particle density distribution functions, which is coherently discretized from Boltzmann equation [6] in phase space with certain approximations of the collision term. One of the most popular model is so called BGK (Bhatnagar–Gross–Krook) approximation [7] that models molecular collision as a local relaxation process from a non-equilibrium state to an equilibrium state. Through Chapman–Enskog technique [8], it can be rigorously proved [9] that the lattice Boltzmann equations recover NS equations in the incompressible limit. The most attractive features of LBM are the ease to deal with complicated porous media structure, the capability to integrate additional physical process, complex flow [10,11], computer graphic [12,13], and biomedical simulation [14,15], into a unified computational platform, and the suitability to perform the cutting-edge GPU parallelization [16–21] to achieve revolutionarily speed-up.

* Corresponding authors at: Mechanical Engineering Department, Indiana University-Purdue University Indianapolis, IN 46202, USA (H. (Whitney) Yu).
E-mail addresses: zwang22@kent.edu (Z. Wang), zhao@kent.edu (Y. Zhao), whyu@iupui.edu (H. (Whitney) Yu).

http://dx.doi.org/10.1016/j.compfluid.2015.04.004
0045-7930/

The conventional LBM is node based. Fluid particles are sitting at lattice nodes. The particle distribution functions represent the particle density distributions associated with discrete molecular velocities. In the time evolution, particles collide at the nodes and then stream to their prescribed finite neighboring nodes. During the streaming step, a bounce-back scheme and its extensions are applied to deal with boundary condition [10,15,16]. To deal with a curved boundary that cuts the lattice off nodes, node-based LBM uses either point-wise particle density distribution interpolation or particle density distribution transformation into local curve-linear coordinate systems. As a result, most of these computational schemes do not guarantee the conservations of mass and momentum and might fail to maintain the detailed balance among particle density distributions, may resulting in numerical artifacts may contaminate the physics of fluid dynamics [4]. Moreover, the realization of high-order interpolations involving nonlocal information is inefficient in parallel computing. Algorithmically, since the collision step only based on local data, in LBM, the streaming step which involves shifting data to adjacent locations is core to parallel performance [22]. However, to deal with different type of nodes (fluid node or boundary node) in streaming step, it need to check each streaming cell if it is available and perform different treatments to fluid nodes and boundary nodes. Then, massive *if* branches are generated and also cause uncoalesced memory access, it would drop the performance of GPU significantly. This is because modern GPUs are based on Single Instruction, Multiple Data (SIMD) architecture and has limited control units. This feature renders the GPU suitable for performing same task across the entire dataset where little control is needed [23]. The performance would be worse especially for simulating fluid dynamic under complicated biomechanical structure. For example, the simulation of hemodynamics in arteries usually has the number of boundary cells more than 1/5 of total computing domain.

In order to deal with arbitrary curved boundaries, which may move willfully, with physical accuracy and implemental convenience, we have recently developed a volumetric representation of LBM [24] which has served as the major component of a unified mesoscale modeling [25] for patient-specific computational hemodynamics from radiological images to *in vivo* fluid dynamics in blood arteries. The Volumetric Lattice Boltzmann Method (VLBM) regards fluid particles are uniformly distributed in cells and distinguishes fluid, solid, and boundary cells by introducing a parameter P defining the percentage of solid volume in each cell. The major advantage of VLBM is the self-regulation of the parameter P in the volumetric lattice Boltzmann equation (VLBE) for particle collision and streaming. There is no spatial interpolation when dealing with an arbitrarily curved boundary with or without motion. First, the VLBE satisfies mass conservation strictly. Second, the implementation of VLBM is rather simply after the solid volume percentages are determined in boundary cells. And third, no-slip boundary condition is integrated in the streaming formulation thus significantly enhances the capability of parallelization.

In this paper, we perform GPU parallelization for VLBM using a uniform computing scheme for both fluid and boundary cells. The number of program branches inside GPU kernel is minimal leading to fast GPU kernel execution. Optimization of coalesced access of GPU memory is pursued by applying Structure of Arrays (SoA) format to store distribution functions, instead of Arrays of Structure (AoS) which is preferable for serial CPU implementation. In VLBM, time advancing information are purely from the nearest neighboring cells as no spatial interpretation is involved to deal boundary condition. We further avoid replication computation and race condition by using an intermediate variable thus two GPU kernels separately deal with collision and streaming operations. We apply the GPU accelerated VLBM for a pulsatile blood flow in a patient-specific carotid artery segmented from an anonymous clinical CT angiograph image and more than 30 times speedup over the serial counterpart. Simulations of fluid dynamics in three refined meshes beyond the image resolution are performed.

The reminder of the paper is organized as follow. The formulation of VLBM and its GPU parallelization including implementation and optimization strategies are presented in Section 2. In Section 3, an application study of pulsatile blood flow in patient-specific carotid artery is performed. Finally, Section 4 provides a summary discussion and concludes the paper.

2. Volumetric LBM and GPU parallelization

The details of the VLBM are referred to reference [24]. Here we include the main concept and involving equations for comprehension and consistency. In VLBM, the fluid particles are uniformly distributed in lattice cells, as opposed to sitting at lattice nodes in node-based LBM. As shown in Fig. 1, the solid black curve depicts a boundary separating fluid (without dots) and solid (with dots) domains. Cells adjacent to the boundary may be occupied entirely by either solid or fluid, while others may have partial fluid and partial solid volumes. In the whole domain, cells can be categorized through the occupation of solid volume in the cell, defined by $P(\boldsymbol{x},t) = V_S(\boldsymbol{x},t)/V(\boldsymbol{x},t)$ in the lattice cell $\boldsymbol{x}$ at time t, where $V(\boldsymbol{x},t)$ is one cell's volume and $V_S(\boldsymbol{x},t)$ is the volume of solid in the cell. Three distinct cell types are illustrated in Fig. 1: fluid cell ($P = 0$), solid cell ($P = 1$), and boundary cell ($0 < P < 1$).

We introduce function $n_i(\boldsymbol{x},t)$ representing particle distribution function with velocity $\boldsymbol{e}_i$ occupying a lattice cell $\boldsymbol{x}$ at time t and deal with the time evolution of particle distribution function (PDF), referred to VLBE,

$$n_i(\boldsymbol{x}+\boldsymbol{e}_i\Delta t, t+\Delta t) = n_i(\boldsymbol{x},t) - \frac{1}{\tau}\left(n_i(\boldsymbol{x},t) - n_i^{eq}(\boldsymbol{x},t)\right) \tag{1}$$

where $i = 0,1,\ldots,b$ is the predefined directions of molecular motion and τ is relaxation time. The equilibrium PDF $n_i^{eq}(x,t)$ is formulated as:

$$n_i^{eq}(x,t) = N\omega_i\left(1 + \frac{3\boldsymbol{e}_i\cdot\boldsymbol{u}}{c^2} + \frac{9(\boldsymbol{e}_i\cdot\boldsymbol{u})^2}{2c^4} - \frac{3\boldsymbol{u}\cdot\boldsymbol{u}}{2c^2}\right) \tag{2}$$

with $N(x,t) = \sum_{i=0}^{b} n_i(x,t)$, ω_i the weighting factor of the ith direction, and c lattice speed (in practice it usually set to 1). In the VLBE, the number of discrete direction of molecular motion b, the discrete molecular velocity $\boldsymbol{e}_i$, and the weighing factor ω_i are determined by selected lattice models. Fig. 1 provides an example for the

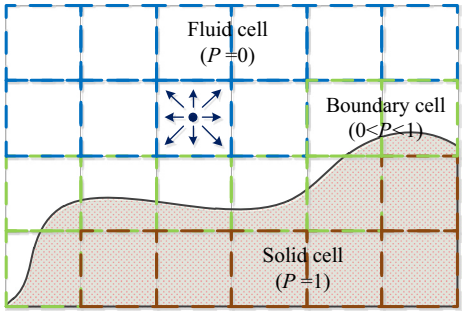


Fig. 1. Volumetric representation of LBM. An arbitrary curve (black line) separates the field into fluid and solid at top (without dots) and bottom (with dots) respectively. Three types of cells are fluid cell ($P = 0$, empty square with blue dashed lines), solid cell ($P = 1$, dotted square with Brown solid line), and boundary cell ($0 < P < 1$, red-filled square with Green dashed lines). Fluid particles are uniformly distributed in fluid and boundary cells. n_i ($(i = 0,1,\ldots,8)$ are particle distribution functions in D2Q9 lattice model as an example. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

discrete PDFs n_i $(i = 0, 1, \ldots, 8)$ on a 2-D lattice with 9 directions called D2Q9 lattice model.

It is noted that the relation between a particle density distribution function f_i in conventional node based LBM and a PDF n_i in the VLBM is as follows:

$$f_i(\boldsymbol{x}, t) = n_i(\boldsymbol{x}, t)/(1 - P(\boldsymbol{x}, t)) \tag{3}$$

where $0 \leqslant P < 1$. For fluid cells ($P = 0$), the values of n_i and f_i are identical for taking the total volume of a cell unity.

The particle number and velocity are obtained as follows

$$N(\boldsymbol{x}, t) = \sum_{i=0}^{b} n_i(\boldsymbol{x}, t) \tag{4}$$

$$u(\boldsymbol{x}, t) = \sum_{i=0}^{b} \boldsymbol{e}_i n_i(\boldsymbol{x}, t)/N(\boldsymbol{x}, t) \tag{5}$$

The density and the pressure can be obtained from $\rho(\boldsymbol{x}, t) = N(\boldsymbol{x}, t)/(1 - P(\boldsymbol{x}, t))$, $(0 \leqslant P < 1)$ and $p(\boldsymbol{x}, t) - p_0 = c_s^2[\rho(\boldsymbol{x}, t) - \rho_0]$ respectively where p_0 and ρ_0 are reference pressure and density respectively. The VLBM can handle arbitrary boundary orientation with respect to the mesh and it satisfies mass conservation strictly [24].

The VLBEs [24] are characterized and self-regularized by the volumetric function $P(\boldsymbol{x}, t)$. Thus computing $P(\boldsymbol{x}, t)$ is the key to employ VLBM. For patient-specific computational hemodynamics, image segmentation from medical CT/MRI images is the necessary by soling a level set equation that tracks an evolving surface Γ through a signed distance field $\phi(\boldsymbol{x}, t)$ to the surface: $\phi(\boldsymbol{x}, t) > 0$ if $\phi(\boldsymbol{x}, t)$ is outside of Γ, $\phi(\boldsymbol{x}, t) < 0$ if $\phi(\boldsymbol{x}, t)$ is inside of Γ, and $\phi(\boldsymbol{x}, t) = 0$ if $\phi(\boldsymbol{x}, t)$ is on Γ. After the flow boundary Γ is segmented together with the solved distance field, $P(\boldsymbol{x}, t)$ can be easily computed based on the sign of the distance function $\phi(\boldsymbol{x}, t)$. For a 3-D cell with 8 mesh nodes, if all the ϕs are negative, the cell is inside the flow domain thus $P = 0$ while all the ϕs are negative, the cell is outside the flow domain thus $P = 1$. Otherwise, the cell is a boundary cell occupied by partial solid and partial fluid. To compute the solid occupation in a boundary cell, we uniformly divide the cell into a refined mesh with q^3 small cubes. The ϕ values at the 8 nodes of each cube are interpolated from the ϕ values of the boundary cell. For cube i located at $\boldsymbol{x}_c$, if the value $\phi(x_c)$ is positive, implying the cube is outside of the flow domain, we set $V_i(\boldsymbol{x}_c) = 1$. Otherwise, $V_i(\boldsymbol{x}_c) = 0$. The total solid volume of the boundary cell $P = \sum_{i=1}^{q^3} V_i/q^3$. The details about the anatomical segmentation and boundary information extraction will be presented in [25].

2.1. VLBM parallel algorithm

The time evolution of VLBE, Eq. (1), can be divided into two operations, collision and streaming, which are self-regularized through the volumetric parameter $P(\boldsymbol{x}, t)$ specifically taking into account the arbitrary boundary. Since the presentation of PDF $n_i(\boldsymbol{x}, t)$ integrate fluid cell and boundary cell, we can perform same computation to these cells to minimize program branches and therefore to improve parallelism. To avoid replication computation and race condition, we introduce two GPU kernels to compute each step. And each kernel is parallelized such that one GPU thread is mapped to one cell in the computing domain.

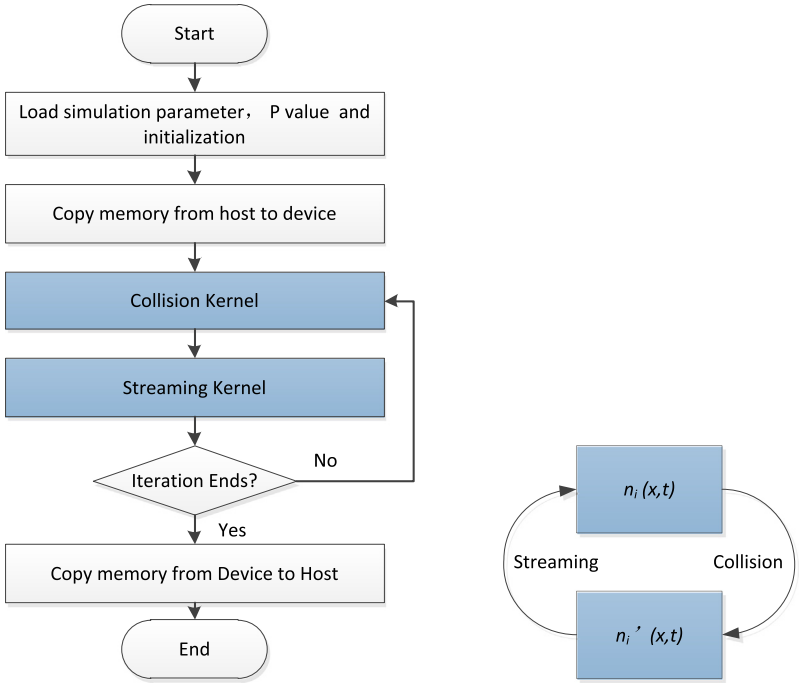
2.1.1. Collision kernel

In order to compute equilibrium PDF $n_i^{eq}(\boldsymbol{x}, t)$, first we update the particle number $N(\boldsymbol{x}, t)$ and velocity $\mathbf{u}(\boldsymbol{x}, t)$ as Eqs. (4) and (5). Then, compute $n_i^{eq}(\boldsymbol{x}, t)$ according to Eq. (2).

Next, we utilize an intermediate variable $n_i'(\boldsymbol{x}, t)$ to save as a 'post-collision' PDF:

$$n_i'(\boldsymbol{x}, t) = n_i(\boldsymbol{x}, t) - \frac{1}{\tau}(n_i(\boldsymbol{x}, t) - n_i^{eq}(\boldsymbol{x}, t)) \tag{6}$$

To avoid memory race condition, $n_i'(\boldsymbol{x}, t)$ is used during the streaming step to store the PDF without overwriting $n_i(\boldsymbol{x}, t)$ as shown in Fig. 2(b).



(a) VLBM workflow (b) Iteration loop in VLBM

Fig. 2. Flow chart of GPU implementation for VLBM. (a) The whole VLBM workflow. (b) The iteration loop in VLBM with $n_i'(\boldsymbol{x}, t)$, it avoids memory race condition in Eq. (1).

Note that in this operation, the computations of Eqs. (2) and 4–6 only involve the current cell $\boldsymbol{x}$ and no information from the neighboring cells are required. Therefore, there is no uncoalesced memory access in GPU, which can yield high efficiency parallel execution.

2.1.2. Streaming kernel

In general, streaming operation results in particle movement from current cell to its neighboring cells. There are two methods which can be described as '*push*' and '*pull*' algorithms depending on the streaming direction. As shown in Fig. 3 '*Push*' approach pushes particles in current cell to its neighbors. In implement, it yields aligned read and unaligned write in GPU memory. Whereas '*Pull*' approach pulls fluid particles from neighboring cells to the current cell, yielding unaligned read and aligned write as shown in Fig. 4. It has been reported that the cost of a misaligned read to be less than a misaligned write in GPU [19,22]. Therefore, we employ pull algorithm in our implementation.

In VLBM, bounce-back boundary condition has been integrated in the streaming operation as

$$n_i(\boldsymbol{x}, t+\Delta t) = (1 - P(x,t))n_i'(\boldsymbol{x} - \boldsymbol{e}_i\Delta t, t) + P(\boldsymbol{x} + \boldsymbol{e}_{i^*}\Delta t, t)n_{i^*}'(\boldsymbol{x}, t) \tag{7}$$

where i^* corresponds to the direction opposite to the ith direction $\boldsymbol{e}_{i^*} = \mathbf{e}_{-i}$. Particles in cell $\boldsymbol{x}$ at time $\mathbf{t} + \Delta\mathbf{t}$ in a streaming operation are from two sources: (i) streaming from its upwind neighboring cells, $(1 - P(\boldsymbol{x}, t))n_i'(\boldsymbol{x} - \boldsymbol{e}_i\Delta t, t)$ and (ii) bounce-back from the downwind cells, $P(\boldsymbol{x} + \boldsymbol{e}_{i^*}\Delta t, t)n_{i^*}'(\boldsymbol{x}, t)$. This modified streaming process ensures that particles are advected or reflected to their appropriate cells in the fluid domain without mass loss. When $P(\boldsymbol{x}, t) = 0$, Eq. (7) recovers the streaming operation in conventional node-based LBM where only upwind streaming occurs. And when $P(\boldsymbol{x}, t) = 1$, Eq. (7) yields standard bounce-back boundary condition.

In conventional LBM parallel implementation, collision and streaming operations are usually blended into one GPU kernel to reduce the number of memory access [17–19]. After the kernel is done, $n_i'(\boldsymbol{x}, t)$ and $n_i(\boldsymbol{x}, t)$ are switched. If same GPU kernel is employed in VLBM, there will involves a replicated computation as two requests of $n_i'(\boldsymbol{x}, t)$ are involved, as seen in Eq. (7), in the streaming operation of VLBM. By separating the collision and streaming kernels, we can not only avoid the replication but also save one switch operation. Meanwhile, the separation of collision and streaming kernels can avoid memory race condition, as thread synchronization implicitly occurs when each kernel is performed. Moreover, the separation makes the implementation convenient to include more complicate collision model (e.g. turbulence model or extra body force terms) and boundary conditions.

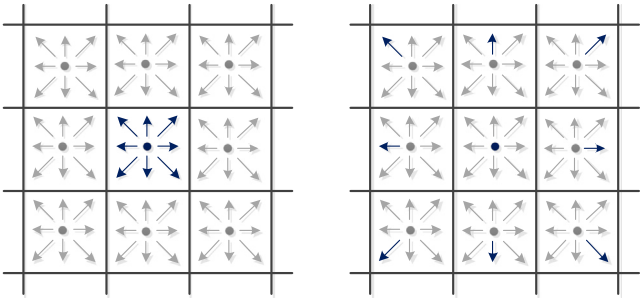


Fig. 3. Push streaming scheme, current cell's PDFs which are represented by blue arrows are propagate to its neighbors in next time step. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

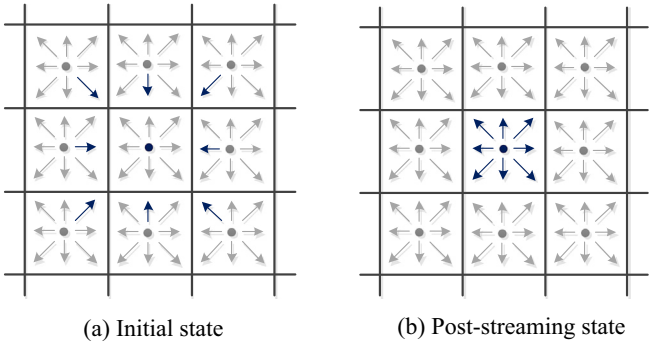


Fig. 4. Pull streaming scheme, the updated PDFs of previous time step in neighbors which represented by blue arrows are propagate to the current cell. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2.2. Memory arrangement

We take D3Q19 lattice model, shown in Fig. 5, as an example to show the memory arrangement strategy. Assuming the size of a computation domain is $Nx * Ny * Nz$. In whole computing process, we need locate $Nx * Ny * Nz$ globe memory in GPU for $P(\boldsymbol{x}, t)$ to store volume information, and separately locate $19 * Nx * Ny * Nz$ memory for both $n_i(\boldsymbol{x}, t)$ and $n_i'(\boldsymbol{x}, t)$. In collision kernel, some intermediary results (like, macroscopic value density $N(\boldsymbol{x}, t)$, $\boldsymbol{u}(\boldsymbol{x}, t)$ and equilibria $n_i^{eq}(\boldsymbol{x}, t)$) are saved in a register for each thread, as these variables are defined locally in a kernel. The registers are a set of very fast, low latency memories. Its transfer speed is ten times faster than globe memory, but only available in limited quantity. To increase the memory access performance, we next optimize the globe memory access pattern and register distribution.

An efficient way to reduce the memory latency of a GPU program is to ensure that memory accesses are coalesced, meaning all the threads in a block access consecutive memory locations. Thus, the memory accesses are combined into one single request by the hardware. In globe memory, to avoid extra de-referencing, it is a common practice to flatten multiple dimension arrays into a single dimension. Regularly, on the CPU, the PDF n_i is stored as in a 4-D array such as $n[z][y][x][i]$. Its flatten form is $n[z * Nx * Ny * 19 + y * Nx * 19 + x * 19 + i]$ for D3Q19 model. This format is referred as 'Arrays of Structure' (AoS), which corresponds to a single array with one element per spatial location, and each element store 19 components of n_i as shown Fig. 6. It has been show that SoA is preferable for serial CPU implementations, as the main thread would access the nodes one by one, and n_i would be likely to be stored in fast cache in CPU with this pattern. While in GPU, such an access pattern is uncoalesced since the threads are not accessing to consecutive memory locations. For example, the accesses to the distribution n_0 by each thread are separated in memory space by $19 * 4 = 76$ bytes (the 19 distributions n_i) and cannot be grouped into a single big memory access. They need to

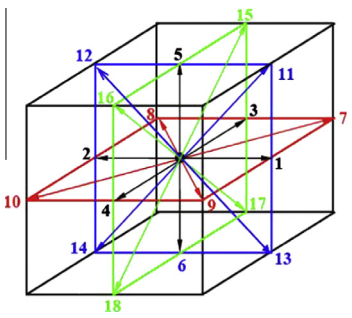


Fig. 5. D3Q19 lattice model.

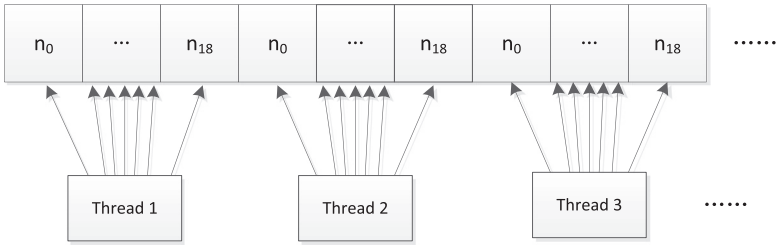


Fig. 6. Memory access pattern for an Array of Structures, that $n_i(\boldsymbol{x},t)$ is addressed as $n[z*Nx*Ny*19+y*Nx*19+x*19+i]$.

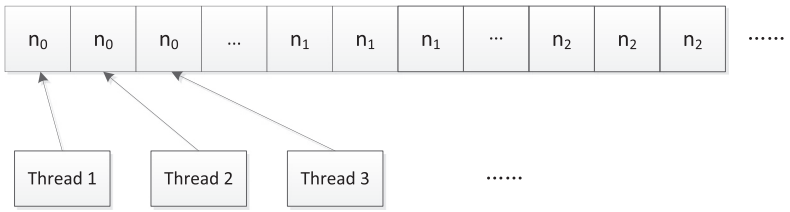


Fig. 7. Memory access pattern for an Structure of Array, Memory access pattern for an Array of Structures, that $n_i(\boldsymbol{x},t)$ is addressed as $n[i*Nx*Ny*Nz+z*Ny*Nx+y*Nx+x]$.

be serialized into a lot of small accesses, which significantly slows down the execution of the kernel.

We employ another alternate ordering method for $n_i(\boldsymbol{x},t)$ which is so called 'Structures of Array' (SoA) format, as shown in Fig. 7. The set of 19 distribution functions of $n_i(\boldsymbol{x},t)$ are stored without interlace. In this case, $n_i(\boldsymbol{x},t)$ is addressed as $n[i*Nx*Ny*Nz+z*Ny*Nx+y*Nx+x]$. Storing n in order of i and then by spatial coordinates will lead neighboring threads within one warp (typically, 32 threads) to access consecutive memory. Then, the data transactions can be grouped into a single larger transaction resulting in a coalesced access to improve the put of globe memory [19,23].

For the existing second generation Kepler GPUs (compute capability 3.5), a single Streaming Multiprocessors (SM) contains 65,536 registers. The D3Q19 solver needs approximately 50 registers per thread. If a block size of 1024 were used in GPU kernel, a total of about 50K threads would be needed for each block, and thus only one block could be launched, causing the remaining 15K registers unused. If a block size of 256 threads is used, it can load 6 blocks for each SM at one time Furthermore, when a kernel is allocated in a GPU, the maximum number of threads in an SM is from 768 to 2048, depending on the compute capability. For maximum compatibility, 256 threads per block can get 100% utilization across all levels of the hardware.

To avoid poor memory coalescing, we arrange threads to match the layout of memory. The following Cuda C code snippet is an example to summarize the arrangement of memory and thread for computing streaming step as shown in Listing 1.

It is noted that some existing work use shared memory and an intrinsic memory-less intra-warp shuffle operation for the memory intensive streaming operations [26,27]. However, it have been shown to be ineffective at improving the performance of the memory-intensive streaming operation in current Kepler GPU, in spite of the fact that it increases the number of coalesced accesses to globe memory [22]. Therefore, a 'naive' implementation that has misaligned access to globe memory is taken due to its lower register usage and no need for any additional control flow.

3. Application study

With the accelerated VLBM, we perform patient-specific computational hemodynamics for an anonymous diseased carotid artery anatomically segmented from a CT angiography taken during a former clinical visit. The original image resolution is $0.4*0.4*0.45$ (mm^3) and the image size is $512*512*52$($pixel^3$). The image segmentation is done by simplified LBM [25], which solves a level set equation. The VLBM resolution from the image is $69*64*176$. In order to check the convergence, we generated three larger resolutions of $88*81*228$, $88*81*228$, and $108*101*286$ through interpolation during the segmentation. Convergence check has confirmed that the resolution based on the original image information is good. At the artery inlet, a pulsatile velocity from a generic ultrasound record, seen in Fig. 8(A). The entire pulsation is divided by 1008 time points to be fed in. At each time point, a paraboloid-like 2D velocity field [25] with the given velocity value as the maximum at center and zero at wall to mimic the realistic blood flow.

The computation is carried out on an NVIDIA Geforce GTX 780 GPU, which has 2304 CUDA cores with 900 MHZ clock frequency and 3 GB globe memory. To evaluate the GPU parallel performance, the serial computation is performed with Intel i7-3770 CPU, which has 4 computing cores with 3.4 GHZ and 8 GB DDR3 Random Access Memory (RAM). For the carotid artery study, the VLBM blood flow simulation is evaluated for 5000 iteration steps and four grid resolutions with different ratios between boundary cells and fluid cells were considered and Table 1 shows the execution times for each cases.

As the resolution increase, the number of boundary cell which is related to surface area of vessel would increase more slowly than the number of fluid cell relating to volume. That makes the ratios between boundary cells and fluid cells reduce from 1/4 to 1/7. But the performance of parallel computing is robust to that changes with about 36 times speedup, due to perform same computation scheme to both boundary cells and fluid cells. On the other hand, different treatment need be performed to deal with inlet and outlet condition, which would cause execution paths in streaming step. But since relatively few cell in source and sink slices, the aspect is not crucial for the overall performance. The GPU acceleration reduces the computation cost of highest resolution from two days (serial VLBM) to one hour (GPU VLBM). For original image resolution, CPU 2 h, GPU 3.4 min for just one pulsatile cycle which need 31,705 iterations. This acceleration is extraordinarily significant since it does not require to use remote supercomputing resources.

```
__global__ void streaming (float *n1, float *n2, float *Pvalue)
{
 int x = blockIdx.x*blockDim.x + threadIdx.x;
 int y = blockIdx.y*blockDim.y + threadIdx.y;
 int z = blockIdx.z*blockDim.z + threadIdx.z;
 int index= z*Nx*Ny+y*Nx+x;
 int size = Nx*Ny*Nz;
 //n_i(x,t) is addressed as
 // n [i * Nx * Ny * Nz +z * Ny * Nx + y * Nx + x].
 n1[index] = n2[index];
 n1[index + 1*size] = (1-Pvalue[index]) *
    n2[(z)*Nx*Ny+(y)*Nx+(x-1)+1*size] +
    Pvalue[(z) *Nx*Ny+(y)* Nx+(x-1)]*n2[index+2*size];
 ......
n1[index + 18*size] = (1-Pvalue[index]) *
  n2[(z+1)*Nx*Ny+(y+1)*Nz+(x)+18*size] +
  Pvalue[(z+1)*Nx*Ny+(y+1)*Nz+(x)]*n2[index+15*size];
}

LBM_streaming(float *n1, float *n2, float *Pvalue)
{
 dim3 blockSize(16,16,1);      // 256(16*16) threads per block
// Arrange threads to match the layout of memory
 dim3 gridSize(Nx/ blockSize.x,
               Ny/ blockSize.y, Nz/ blockSize.z);
 streaming <<<gridSize, blockSize>>> (n1, n2, Pvalue);
}
```

Listing 1. Thread and memory arrangement for streaming step.

Next, we present physical results of pulsatile flow is the carotid artery where a stenosis is present in the internal carotid artery (ICA, left branch) near the bifurcation. The representative time points include two, (a) and (b), during acceleration (systole), three, (d)–(f), during deceleration (diastole), as well as the peak point (c). To compare the behavior in acceleration and deceleration, we have selected same *Re* numbers, *Re* 200 for (a) and (e) and *Re* = 400 for

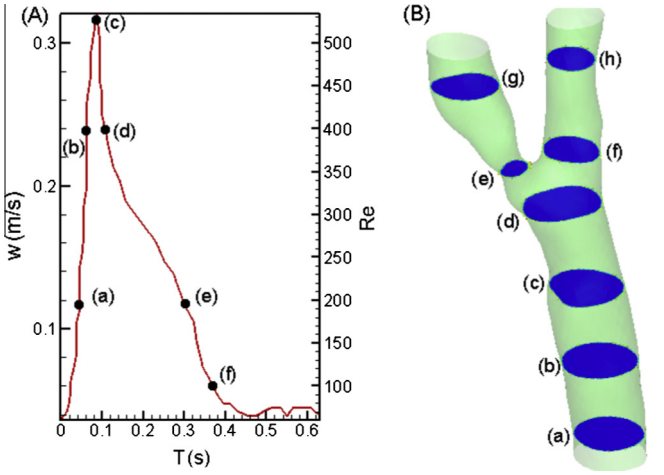


Fig. 8. (A). Pulsatile velocity profile extracted from a generic Doppler Ultrasound recording to drive blood in the carotid artery at the inlet. (B). Representative planes where quantitative velocity profiles are studied.

Table 1
Execution times of blood simulation for different grid size.

Grid size	Resolution (m/cell)	Boundary / fluid	CPU (s)	GPU (s)	Speed up
69 × 64 × 176	2.225e−4	~1/4	1199	34	35.2
86 × 79 × 221	1.7660e−4	~1/5	2255	63	35.8
106 × 99 × 279	1.4017e−4	~1/6	4183	117	35.7
133 × 125 × 352	1.1125e−4	~1/7	8151	226	36.1

(b) and (d). We collect each pair next to each other in the discussion of the results unless otherwise indicated. It is noted that the validation of VLBM has been performed extensively in our previous works [24,25] and the outputs from GPU-VLBM has been confirmed to be identical to those from serial-VLBM, here we present the simulation results without further validations.

Fig. 9 shows the contours of normalized velocity magnitude on represented transverse planes at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A). The contour scales are specified for *Re* = 100 of (f), *Re* = 200 of (a) and (e), *Re* = 400 of (b) and (d), and *Re* = 533 of (c). As expected, larger *Re* numbers (bottom row) driven faster blood flow than the small *Re* numbers (top row). Velocity skewness [28,29] is observed in the ICA at each time point, more severe in diastole, (c)–(e), than systole, (a) and (b). Quantitative measurements of the difference between diastole and systole periods are shown in Fig. 10 where streamwise velocity profiles along an estimated center line on each representative transverse plane, (a)–(f) in Fig. 8(B) at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A). The solid lines are for the plane below the bifurcation and dashed lines are for the top branches. It is seen that in the common carotid artery (CCA) which is the part below the bifurcation and the external carotid artery (ECA, the right branch), velocity profiles are parabolic for all the time points implying that for the intensity of pulsation used in this study, blood flow in both CCA and ECA can be regarded as a Poiseuille flow. Whereas in ICA, (e) and (g), flow is irregular. For the case of (e) where the stenosis is located, the velocity profile (green dashed lines) is irregular with much larger peak value than all other peak values for all the time points, meaning that high velocity maintains at plane (e) during the pulsation. The detail velocity contours on plane (e) at each time point are shown in Fig. 11. While the velocity value varies with the *Re* number, the irregularity (skewness) are more profound during diastole ((d)–(f)).

Wall shear stress (WSS) [25] is of great interest from medical point view as it characterizes the interaction between a blood flow and inner wall of an artery. We show both anterior and posterior views of the WSS distributions on the inner wall of the carotid artery in Figs. 12 and 13 respectively. The contour scales are the same for all the time points. Although the streamwise velocity at the stenosis significantly varies with *Re* number during the pulsation as shown above, the large WSS (velocity gradient) maintains with the same intensity where the stenosis locates for all the time points. When *Re* is higher (bottom rows), the intensive WSS affects larger area in the ICA than low Res (top rows). This implies that when a stenosis is formed, the inner wall may be damaged more profoundly than the inner wall of a normal artery.

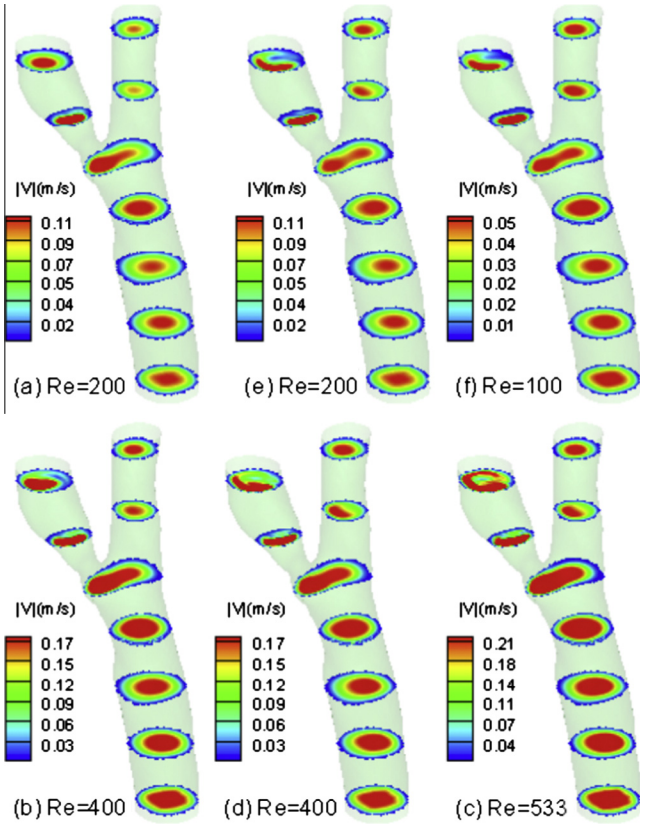


Fig. 9. Contours of normalized velocity magnitude on representative transverse planes at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A).

4. Discussion and future work

In this work, we perform GPU parallelization and optimization for VLBM, which was recently developed for solving complex flow with arbitrary curved boundaries. The advantages of VLBM over conventional node-based LBM stem from the self-regulation of the volumetric parameter $P(x,t)$ in the VLBE for particle collision and streaming. First, the VLBE satisfies mass conservation strictly. Second, the implementation of VLBM is rather simply after the solid volume percentages are determined in boundary cells. And third, no-slip boundary condition is integrated in the streaming formulation thus significantly enhances the capability of parallelization. With no intention to produce results with medical insights, we study a pulsatile blood flow in a patient-specific carotid artery segmented from an anonymous clinical CT image to evaluate the GPU implementation. Above 30 times speed-up is achieved thus make it possible to complete a patient-specific computational hemodynamic simulation within clinical accepted time frame, e.g. less than one hour. Simulations of fluid dynamics and WSS are presented and known behavior of velocity and WSS distributions are captured.

Three types of further work are ongoing. First, we are adding the segmentation part in the GPU platform and further optimizing the GPU algorithm to further increase the acceleration, targeting to complete a typical patient-specific computation from image to quantitative fluid dynamics within 15 min. Second, we are adding

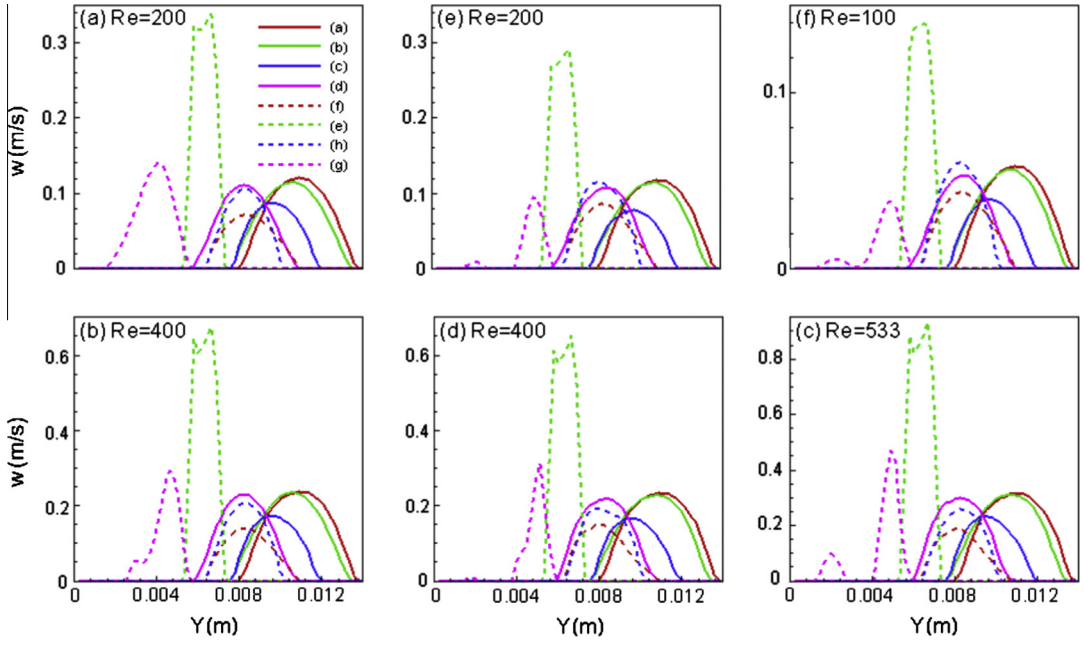


Fig. 10. Streamwise velocity profiles along an estimated center line on each representative transverse plane, (a–f) in Fig. 8(B) at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A).

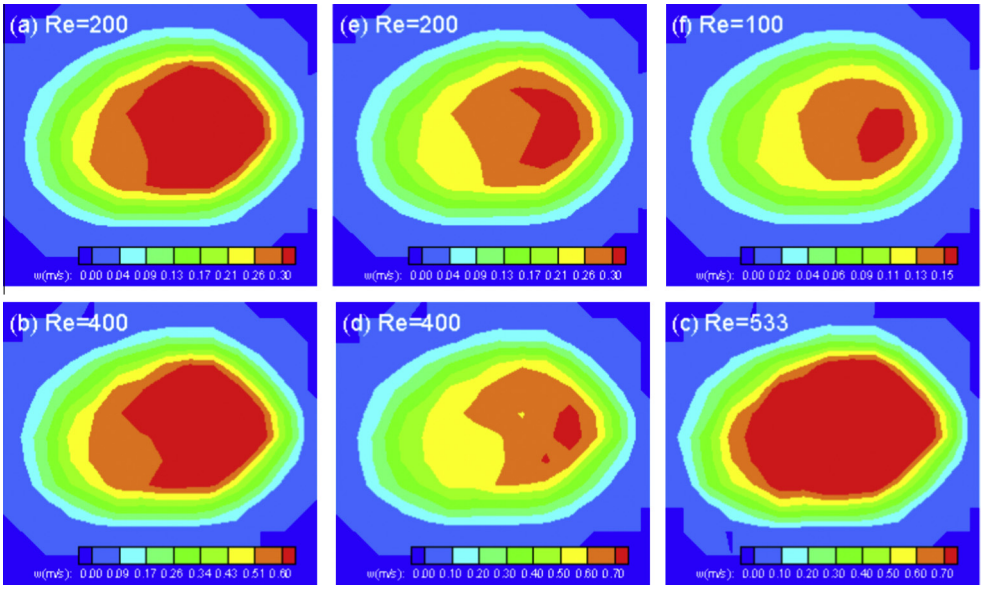


Fig. 11. Contours of streamwise velocity on plane (e) in Fig. 8(B) at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A).

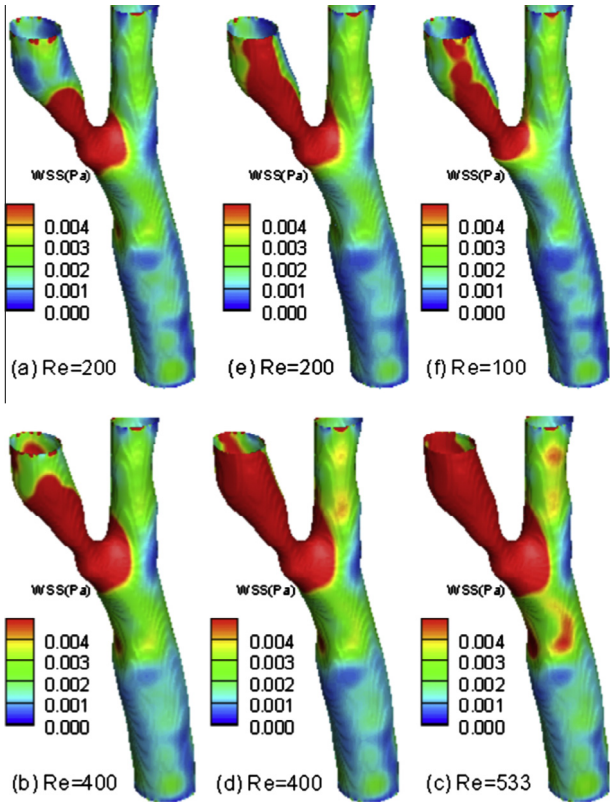


Fig. 12. (Anterior view) WSS distribution on the inner wall at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A).

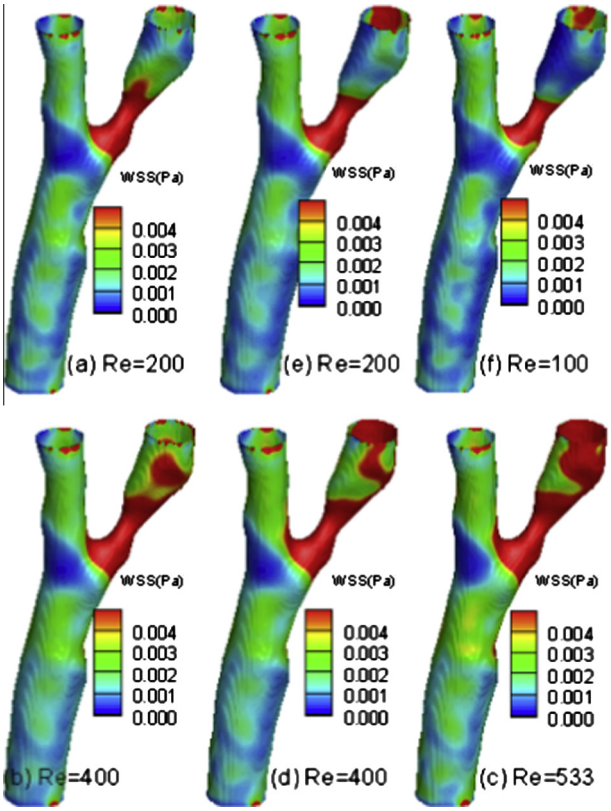


Fig. 13. (Posterior view) WSS distribution on the inner wall at represented time points during a pulsation with indicated *Re* numbers in Fig. 8(A).

one more mesoscale model for deformable structure in the computational platform to mimic the real arteries which are inherently elastic. Third, we are applying the GPU accelerated computational tool to perform secondary analysis of existing medical images from clinic via massive patient-specific computational hemodynamics aimed to identify unprecedented hemodynamic indicators for clinical assessment and prediction of fatal cardiovascular diseases such as stroke and heart attack. Meanwhile, the unified and GPU accelerated computing platform will enable medical practitioners to access the quantitative fluid dynamics and WSS information in diseased arteries simultaneously with CT/MRI imaging promoting deeper understanding of vascular diseases and laying the groundwork for future improvements in patient care and clinical decision making.

References

[1] Chen HD, Chen SY, Matthaeus WH. Recovery of the Navier–Stokes equations using lattice-gas Boltzmann method. Phys Rev A 1992;45:R5339–42.
[2] Benzi R, Succi S, Vergassola M. The lattice Boltzmann-equation—theory and applications. Phys Rep-Rev Sec Phys Lett 1992;222:145–97.
[3] Qian YH, Dhumieres D, Lallemand P. Lattice Boltzmann model for Navier–Stokes equation. Europhys Lett 1992;17:479–84.
[4] Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. Annu Rev Fluid Mech 1997;30:329–64.
[5] Aidun CK, Clausen JR. Lattice-Boltzmann method for complex flows. Annu Rev Fluid Mech 2010;42:439–72.
[6] Harris S. An introduction to the theory of the Boltzmann equation. New York: Holt Rinehart and Winston Inc; 1970.
[7] Bhatnagar PL, Gross EP, Krook M. A model for collisional processes in gases I: small amplitude processes in charged and in neutral one-component systems. Phys Rev 1954;94:511–25.
[8] Chapman S, Cowling TG. The mathematical theory of non-uniform gases. 3rd ed. London: Cambridge University Press; 1970.
[9] He XY, Luo LS. Theory of the lattice Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation. Phys Rev E 1997;56:6811–7.
[10] Ye Yu, Li Kenli. Entropic lattice Boltzmann method based high Reynolds number flow simulation using GUDA on GPU. Comput Fluids 2013;88:241–9.
[11] Myre J, Walsh SDC, Lilja D, Saar MO. Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters. Concur Comput Pract Exp 2011;23:332–50.
[12] Zhao Y. Lattice Boltzmann based PDE solver on the GPU. Visual Comput 2007;24:323–33.
[13] Sun X, Wang Z, Chen G. Parallel active contour with lattice Boltzmann scheme on modern GPU. In: IEEE international conference of image processing; 2012. p. 1709–12.
[14] Chloe A, Tommaso M, Herve D, et al. Lattice Boltzmann method for fast patient-specific simulation of liver tumor ablation from CT images. In: MICCAI; 2013. p. 323–30.
[15] Cosmin Nita, Lucian MI, Constantin S. GPU accelerated blood flow computation using the lattice Boltzmann method. In: High performance extreme computing conference (HPEC); 2013. p. 1–6.
[16] Stratford K, Pagonabarraga I. Parallel simulation of particle suspensions with the lattice Boltzman method. Comput Math Appl 2008;55:1585–93.
[17] Tolke J. Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA. Comput Visual Sci 2010;13:29–39.
[18] Kuznik F, Obrecht C, Rusaouen G, Roux JJ. LBM based flow simulation using GPU computing processor. Comput Math Appl 2010;59:2380–92.
[19] Delbosc N, Summers JL, Khan AI, Kapur N, Noakes CJ. Optimized implementation of the lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation. Comput Math Appl 2014;67:462–75.
[20] Habich J, Feichtinger C, Kostler H, Hager G, Wellein G. Performance engineering for the lattice Boltzmann method on GPGPUs: architectural requirements and performance results. Comput Fluids 2013;80:276–82.
[21] Yu H, Chen R, Wang H, Yuan Z, Zhao Y, An Y, et al. A GPU accelerated lattice Boltzmann simulation for rotational turbulence. Comput Math Appl 2014;67(2):445–51.
[22] Mark JM, Alistair JR. Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture Nvidia GPUs. Comput Phys Commun 2014;185:2566–74.
[23] CUDA C Best Practices Guide v5.5. Nvidia; 2013. p. 3–4.
[24] Yu H, Chen X, Wang Z, Deep D, Lima E, Zhao Y, et al. Mass-conserved volumetric lattice Boltzmann method for complex flows with willfully moving boundaries. Phys Rev E 2014;89:063304.
[25] Yu H, Wang Z, Zhao Y, et al. Unified mesoscale modeling—from radiological images to in vivo fluid dynamics in blood arteries. Int J Numer Meth Biomed Eng 2015. submitted for publication.
[26] Rinaldi P, Dari E, Venere M, Clausse A. A lattice Boltzmann Solver for 3D fluid simulation on GPU. Simul Model Pract Theory 2012;25:163–71.
[27] Astorino M, Sagredo JB, Quarteroni A. A modular lattice Boltzmann solver for GPU computing processors. SeMA J 2013;59:53–7.
[28] Vincent PE, Plata AM, Hunt AAE, Weinberg PD, Sherwin SJ. Blood flow in the rabbit aortic arch and descending thoracic aorta. J R Soc Interface 2011;8:1708–19.
[29] Shahcheraghi N, Dwyer HA, Cheer AY, Barakat AI, Rutaganira T. Unsteady and three-dimensional simulation of blood flow in the human aortic arch. Trans ASME 2002;124(8):378–87.