Computers and Fluids 266 (2023) 106038

Contents lists available at ScienceDirect

Computers and Fluids

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

GPU accelerated volumetric lattice Boltzmann model for image-based hemodynamics in portal hypertension

Bo Shang [a], Rou Chen [a,*], Weiwei Yan [a,*], Huidan(Whitney) Yu [b]

[a] College of Metrology & Measurement Engineering, China Jiliang University, Hangzhou, Zhejiang, 310018, China
[b] Department of Mechanical & Energy Engineering, Indiana University-Purdue University Indianapolis, Indianapolis (IUPUI), IN 46202, USA

ARTICLE INFO

Keywords:
Volumetric lattice Boltzmann method
GPU parallel computing
Hepatic-portal venous system
Image-based hemodynamics

ABSTRACT

We are developing a new technique for monitoring portal hypertension by the pressure gradient between the portal vein and the inferior vena cava (PPG) based on non-invasive measurements (MRI images). Massive parametrization and classification are required to investigate the underlying relationship between the porosity and the stages of liver cirrhosis numerically, and the hepatic-portal venous system is a multi-scale system. Both of them need high computational costs. The suitability of the lattice Boltzmann method for GPU (Graphics Processing Unit) parallel computation provides an opportunity to overcome it. In this paper, we perform GPU parallelization and optimization for the volumetric lattice Boltzmann method with arbitrary geometry based on images. Three application cases, including pipe flow, hemodynamics in the portal venous system, and hemodynamics in a simple hepatic-portal venous system, are employed to prove the method can be applied in the hepatic-portal venous system based on accuracy and efficiency. The reliability of the model is qualitatively validated by the analytical solution of velocity and pressure difference distribution of pipe flow and quantitatively confirmed by the pulsatility of velocity and pressure difference that can be neglected in the portal venous system. The performance of the application cases is examined with Intel Broadwell E5-2683 v3@ 2.30 GHz (CPU) and NVIDIA Tesla V100 16GB (GPU). It shows the GPU algorithm for sparse geometry (SPARSE) has a similar speed to the regular GPU algorithm for dense geometry (DENSE) when the fluid volume fraction (q) is close to 1. And SPARSE speeds up to 2.2 times compared with DENSE when q is in the range of 0.19∼0.27. Meanwhile, the saving ratio of memory cost depends on q. For a numerical case in the hepatic-portal venous system, i.e., a large-scale system, parallel execution can be converged around half an hour with SPARSE, while the memory spills the limitation with DENSE with a single GPU. Hence, multi-GPU implementation is applied to release the limitation, and it can improve performance by increasing the number of GPU cards. In summary, the method presented in the paper is feasibility applied in the hepatic-portal venous system, laying the foundation of the new technique development.

1. Introduction

Portal hypertension is a clinical syndrome defined as increased pressure in the portal venous system, mainly caused by cirrhosis, and responsible for lethal complications such as ascites, gastro-esophageal varices, variceal hemorrhage, and hepatic encephalopathy [1,2]. Especially, variceal bleeding has a mortality of 12%–20% at six weeks and recurs in 2/3 within two years without adequate treatment [3]. The pressure gradient between the portal vein and the inferior vena cava (PPG) qualitatively characterizes portal hypertension. However, it is rarely used in clinical because transhepatic or transvenous catheterization has a high risk of intraperitoneal bleeding [4]. Monitoring hepatic venous pressure gradient (HVPG) is recommended as a gold-standard technique to assess portal hypertension [5]. However, it is limited to invasiveness which is harmful to the examinee. Therefore, there is a critical needing for noninvasive techniques to measure the PPG/HVPG with the characteristic of safe, accurate, and easy to operate. Recently, Qi [6] developed a technique based on a computational hemodynamic model with CTA images. A new indicator, virtual HVPG, was found to correlate significantly with HVPG under moderate portal hypertension. The major limitation is that it is relatively time-consuming, caused by 3D model reconstructing and hemodynamic computing. Hence, we develop a new computational hemodynamic model to access the PPG based on the GPU-accelerated VLBM with MRI images. The main challenge of our development is as follows: (1) Validate the efficiency and accuracy of our technique. (2) Model the liver as porous media with the image information of hepatic and portal venules and build

* Corresponding authors.
E-mail addresses: rouchen@cjlu.edu.cn (R. Chen), yanww@cjlu.edu.cn (W. Yan).

https://doi.org/10.1016/j.compfluid.2023.106038
Received 10 February 2023; Received in revised form 9 July 2023; Accepted 30 July 2023
Available online 23 August 2023
0045-7930/© 2023 Elsevier Ltd. All rights reserved.

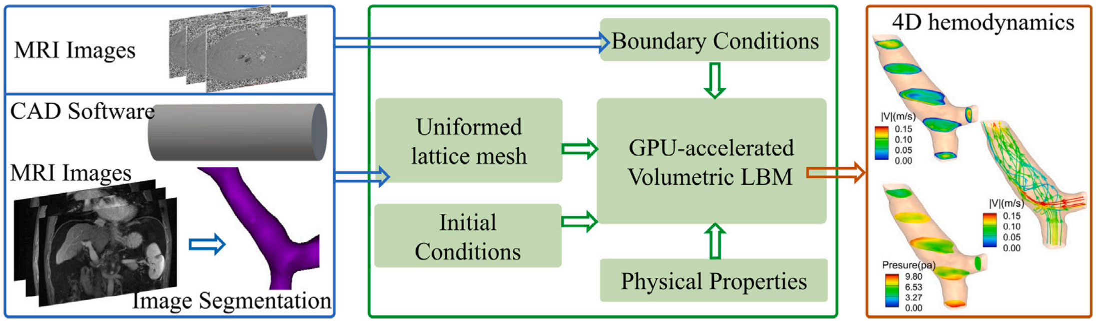

Fig. 1. Flow chart of image-based computational hemodynamics technique.

the correlation between the porosity and the stage of liver cirrhosis through a massive parametric study. Therefore, validating the feasibility of our technique is the preliminary step to developing the new technology that is expected to be a noninvasive and accurate method for evaluating hepatic portal vein pressure. In this paper, we perform GPU parallelization and optimization for VLBM in the hepatic-portal venous system. The model of the liver will be discussed in the future.

Optimization of GPU acceleration for LBM has been developed mainly in memory access patterns, register utilization, and overlap of memory transfer and arithmetic operations. Memory access pattern attracts the most attention in the literature as follows.

Propagation schemes: GPU parallelization of LBM majority parts in each iteration are collision and streaming. Since collision occurs at the local nodes and streaming requires information from neighboring nodes, the efforts on the fast access to the memory have been on the development of propagation schemes [7] for the streaming part. Shift algorithm with shared memory [8,9] was applied at the beginning to avoid misaligned accessing at the expense of adding an extra kernel to exchange data by GeForce 8800 Ultra. The propagation scheme of the A–A pattern (only one set of distribution functions (DFs)) based on the shift algorithm with shared memory was developed [10] in 2009. The A–A pattern reduced GPU memory by 50% compared with the A–B pattern (two sets of DFs in memory). Shortly after, a split propagation scheme (misaligned write) and reversed propagation scheme (misaligned read) [11] were developed, which have a 15% improvement compared with a shared memory scheme with the GTX 295.

Data layout: Two popular schemes include array of structure (AoS) and structure of array (SoA). SoA switches from AoS for the optimization of GPU parallelism [12]. It has a 7–10 times speed increase compared with AoS by Tesla Kepler K20 [13]. Herschlag et al. [14] applied collected SoA (CSoA) in the GPU architectures and accelerated computational speed by 5%–20% compared with the SoA layout.

Memory addressing: Direct addressing scheme is universally employed in the dense computational domain. In the case of complex geometry, the fluid volume fraction in the uniformed lattice domain is small. For example, the portal venous system tested in the paper has a fluid volume fraction that varies from 13% to 27%, i.e., the majority of lattice information is superfluous. Meanwhile, the basis for the feasibility of new technology is a large computational scale with a fine resolution constrained by memory utilization. Therefore, indirect addressing and semi-addressing schemes originally from the approach of CPUs are necessarily applied to complex geometry. Compared with direct address, they provide higher computational speed and lower memory consumption when the fluid volume fractions are less than 0.5 [14]. Herschlage et al. [15] recommend the semi-addressing scheme for complex geometry problems, which have 1.1–1.3 and 1.25–1.5 times acceleration through Tesla P100 and V100, respectively.

Two other aspects need to be addressed. Accessing register memory consumes zero clock cycles per instruction compared to 400–600 cycles for global memory. While the amount of registers is limited. An NVIDIA compiler provides a flag limiting the register usage for register utilization [16]. Meanwhile, two strategies for the overlap of memory transfer and arithmetic operations include tiling the 3D lattice grid into smaller 3D blocks [17] and branch divergence removal [13]. Meanwhile, the development of parallelization combined with GPU, CPU, and network has recently been popular in solving multi-scale, multi-physics research problems. The parallelization contains multi-GPU [18], LBM code with CUDA and OpenMP for multi-GPU clusters [19], LBM with CUDA and MPI [20], and a multi-node MPI/OpenMP LBM code with OpenACC [21].

In this paper, we perform GPU parallelization and optimization for the volumetric lattice Boltzmann method for image-based hemodynamics in portal hypertension. The volumetric lattice Boltzmann method was developed and refined by Yu's group [22,23] and has been verified as a reliable method to noninvasively quantify hemodynamic abnormalities in human arteries [24–26] based on patient medical imaging data. Combining features of GPU with VLBM is significant in achieving an efficient GPU-VLBM algorithm. An A–B (2 sets of distribution functions) pattern, dynamic allocation, SoA, pull propagation scheme, semi-indirect addressing, register utilization, and multi-GPU are employed in this paper. A pipe directly generated from images full of the blood flow in the same order as the portal venous system length scale is applied to validate the method's reliability qualitatively. And the accuracy of the method is quantitatively confirmed by the pulsatility of velocity and pressure difference in the portal venous system. The efficiency of the method is derived by the performance of the model, which is tested by three different application cases, including pipe flow, portal venous system, and portal-hepatic venous system with Intel Broadwell E5-2683 v3@ 2.30 GHz (CPU) and NVIDIA Tesla V100 16 GB (GPU). Finally, a summary discussion is provided.

2. Computational methodology

The flow chart of the image-based computational hemodynamics technique shown in Fig. 1 contains pre-processing, kernel computational hemodynamics, and post-processing. The uniformed lattice mesh, initial boundary, physical properties, and boundary conditions are the pre-processing of the computational simulation. The uniformed lattice mesh is generated from the arbitrary geometry reconstructed from the patient-specific MRI images or extracted from the CAD software. The boundary conditions are derived from the MRI images, which can provide the two-dimensional velocity distribution in specific locations. And the kernel part of computational hemodynamics is based on the GPU-accelerated volumetric lattice Boltzmann method. Finally, the 4D hemodynamical analysis is derived, including three-dimensional spatial and one-dimensional temporal information, which can provide guidelines for the clinical diagnosis. This section presents the methodology developed from the following three aspects: the volumetric lattice Boltzmann method, lattice mesh generation, and GPU parallelism and optimization.

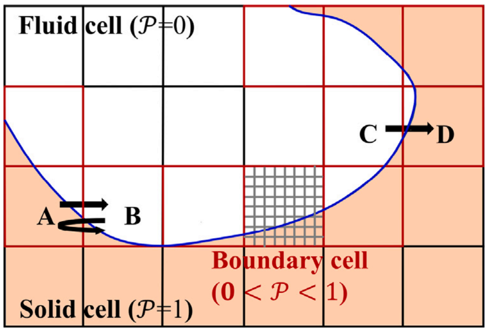

Fig. 2. Illustration of the volumetric parameter $\mathcal{P}$ and two scenarios of bounce-back boundary treatment.

2.1. Volumetric lattice Boltzmann method

The volumetric lattice Boltzmann method is a mesoscopic method for computational fluid dynamics in that fluid particles are uniformly distributed in the lattice cells instead of at the lattice nodes, which is indicated in the traditional lattice Boltzmann method. Except for the inherent advantages of the lattice Boltzmann method, including simple implementation, easy handling of complicated boundaries, and well suited for GPU parallel computing, VLBM improves the efficiency of solving arbitrary geometry with or without moving boundary by introducing a volumetric parameter $\mathcal{P}(\mathbf{x}, t)$ defined as the percentage of the solid cells occupation in each unit lattice cell at the specific time t. Hence, the computational domain can be distinguished by three different types of cells, as Fig. 2 shows solid cells ($\mathcal{P} = 1$), fluid cells ($\mathcal{P} = 0$), and boundary cells $0 < \mathcal{P} < 1$. The detailed description of the method of $\mathcal{P}$ generation, i.e., lattice mesh generation, is shown in Section 2.2. The flow chart of VLBM is the same as traditional LBM, including initial condition, LBM evolution, and boundary condition. Still, the distribution functions are developed with volume-based cells, and the boundary condition of walls is self-regularized by $\mathcal{P}$ and unified with streaming propagation. In the simulation of the hemodynamics in the portal venous system and portal-hepatic venous system, the geometries system is assumed as a fluid domain with rigid walls, i.e., fixed geometries ($\mathcal{P}(\mathbf{x}, t) = \mathcal{P}(\mathbf{x})$) and the blood flow is treated as the incompressible Newtonian flow. The governing equations of fluids with arbitrary geometry, including continuity and Navier–Stokes equations, can be simplified as follows.

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \upsilon \nabla^2 \mathbf{u} \tag{2}$$

where υ is the kinematic viscosity of the fluid. The volumetric lattice Boltzmann equations employed for recovering the governing equation by Chapman–Enskog are synthesized from open Refs. [22,23] showing as follows.

Collision operator: Setting the particle population distribution as $n_\alpha(\mathbf{x}, t)$, the volumetric lattice Boltzmann equation with collision operator indicated by a single relaxation time is

$$n'_\alpha(\mathbf{x}, t) = n_\alpha(\mathbf{x}, t) + \frac{1}{\tau}\left[n_\alpha(\mathbf{x}, t) - n^{eq}_\alpha(\mathbf{x}, t)\right] \tag{3}$$

where $n^{eq}_\alpha(\mathbf{x}, t)$ is equilibrium particle population distribution equation and determined by initial particle numbers per cell $N_0(= 1 - \mathcal{P}(\mathbf{x}))$, lattice speed c, discrete molecular velocities $\mathbf{e}_\alpha$, weighting factors ω_α, and velocity $\mathbf{u}$ as

$$n^{eq}_\alpha(\mathbf{x}, t) = N_0\omega_\alpha\left\{1 + \frac{3\mathbf{e}_\alpha \cdot \mathbf{u}}{c^2} + \frac{9\left(\mathbf{e}_\alpha \cdot \mathbf{u}\right)^2}{2c^4} - \frac{3\mathbf{u} \cdot \mathbf{u}}{2c^2}\right\} \tag{4}$$

$c = \delta x/\delta t = 1$, i.e., $\delta x = \delta t = 1$, meaning that the particles stream one lattice unit per time step. For the three dimensional simulation, D3Q19 model is employed, the weighting factors $\omega_0 = 1/3$, $\omega_{1-6} = 1/18$, $\omega_{7-18} = 1/36$, and the discrete molecular velocities $\mathbf{e}_0 = (0, 0, 0)$, $\mathbf{e}_{1-6} = (\pm1, 0, 0), (0, \pm1, 0), (0, 0, \pm1)$, $\mathbf{e}_{7-18} = (\pm1, \pm1, 0), (\pm1, 0, \pm1), (0, \pm1, \pm1)$.

Streaming propagation: The generalized formulation of streaming propagation reflects the flow of particles from one cell to the neighboring cells, including the fluid and the boundary cells. For the boundary cells, the streaming algorithm is listed in Fig. 2 with two specific scenarios. Assume cells B and D at $\mathbf{x} + \mathbf{e}_\alpha\delta t$ receive particles during streaming from the adjacent cells A and C at position $\mathbf{x}$, respectively. Scenario 1 ($\mathcal{P}_A > \mathcal{P}_B$): the population of particles streaming from A cell into B cell in the $\mathbf{e}_\alpha$ direction contains the particle population from cell A to B(n'_{A_α}) and the bounce-back part of particles that flow from cell B to A in the $\mathbf{e}_{\alpha^*}$ direction($\frac{\mathcal{P}_A - \mathcal{P}_B}{1-\mathcal{P}_B} n'_{B_{\alpha^*}}$, $\alpha^* = -\alpha$). Scenario 2 ($\mathcal{P}_C < \mathcal{P}_D$): cell D receives the particle population from cell C($\frac{1-\mathcal{P}_D}{1-\mathcal{P}_C} n'_{C_\alpha}$) and bounces particle population in the opposite direction($\frac{\mathcal{P}_D - \mathcal{P}_C}{1-\mathcal{P}_C} n'_{C_\alpha}$). Therefore, the generalized streaming propagation is presented as

$$\begin{aligned} n_\alpha(\mathbf{x} + \mathbf{e}_\alpha\delta t, t + \delta t) &= G_\alpha(\mathbf{x} + \mathbf{e}_\alpha\delta t)\frac{1 - \mathcal{P}(\mathbf{x} + \mathbf{e}_\alpha\delta t)}{1 - \mathcal{P}(\mathbf{x})} n'_\alpha(\mathbf{x}, t) + \\ \left[1 - G_\alpha(\mathbf{x} + \mathbf{e}_\alpha\delta t)\right] &\left[n'_\alpha(\mathbf{x}, t) + \frac{\mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{x} + \mathbf{e}_\alpha\delta t)}{1 - \mathcal{P}(\mathbf{x} + \mathbf{e}_\alpha\delta t)} n'_{\alpha^*}(\mathbf{x} + \mathbf{e}_\alpha\delta t, t)\right] \end{aligned} \tag{5}$$

where G is a parameter is introduced to identify the two different scenarios.

$$G_i(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathcal{P}\left(\mathbf{x} + \mathbf{e}_\alpha\delta t\right) \geq \mathcal{P}(\mathbf{x}) \\ 0, & \text{if } \mathcal{P}\left(\mathbf{x} + \mathbf{e}_\alpha\delta t\right) < \mathcal{P}(\mathbf{x}) \end{cases} \tag{6}$$

The density and velocity can be computed by taking zeroth and first moments of particle population distribution function n_α as $\rho(\mathbf{x}, t) = \sum_\alpha n_\alpha(\mathbf{x}, t)/(1 - \mathcal{P}(\mathbf{x}, t))$ and $\mathbf{u}(\mathbf{x}, t) = \sum_\alpha \mathbf{e}_\alpha n_\alpha(\mathbf{x}, t)/\sum_\alpha n_\alpha(\mathbf{x}, t)$, respectively. And the pressure can be derived by $\Delta p(\mathbf{x}, t) = \Delta\rho(\mathbf{x}, t)/3$.

2.2. Lattice mesh generation

For image-based computational hemodynamics, the specific geometry is generated from the image segmentation of medical images and exported in STL format, a standard output format for 3D printing (experimental investigation) and rapid prototyping (visualization). And the lattice mesh for the VLBM simulation is generated by calculating the volumetric parameter $\mathcal{P}(\mathbf{x}, t)$ from STL data. First, read facets orientation and vertex positions of unstructured triangulated mesh and transfer them in the Cartesian coordinate system by the existing open-source packages of READ_stl.m and VOXELSE.m, respectively. Then, generate a signed distance field by calculating the distance function $\phi(\mathbf{x}, t)$ to the geometry surface with the open-source code of ac-reinit.m: $\phi(\mathbf{x}, t)$ is positive, if the position of $\mathbf{x}$ is inside the surface, $\phi(\mathbf{x}, t)$ is negative, if position of $\mathbf{x}$ is outside the surface, and $\phi(\mathbf{x}, t)$=0, if position of $\mathbf{x}$ is on the surface. Finally, the volumetric parameter $\mathcal{P}(\mathbf{x}, t)$ can be computed based on the sign of the distance function $\phi(\mathbf{x}, t)$. Here taking 2-D cells indicated in Fig. 2 as an example to represent how to calculate $\mathcal{P}(\mathbf{x}, t)$. If all four vertex distance functions of a 2-D cell are negative, the 2-D cell is outside the boundary. Hence, $\mathcal{P}(\mathbf{x}, t) = 1$. While all four vertexes with the positive distance function, the 2-D cell is inside the boundary. Thus, $\mathcal{P}(\mathbf{x}, t) = 0$. Otherwise, the four vertexes distance function of the 2-D cell consists of partial inside and partial outside the boundary. The 2-D cell needs to be clarified by refined mesh through interpolation. Assume the 2-D cell is divided into q^2 mesh cells if the distance function of the mesh cell is negative, $V^i_s = 1$, $i = 1, 2, \ldots, q^2$. Otherwise, $V_s = 0$. Hence, $\mathcal{P}(\mathbf{x}, t)$ is obtained by $\mathcal{P}(\mathbf{x}, t) = \sum_{i=1}^{q^2} V^i_s/q^2$. 8^3 mesh cells are usually applied in the 3D simulation.

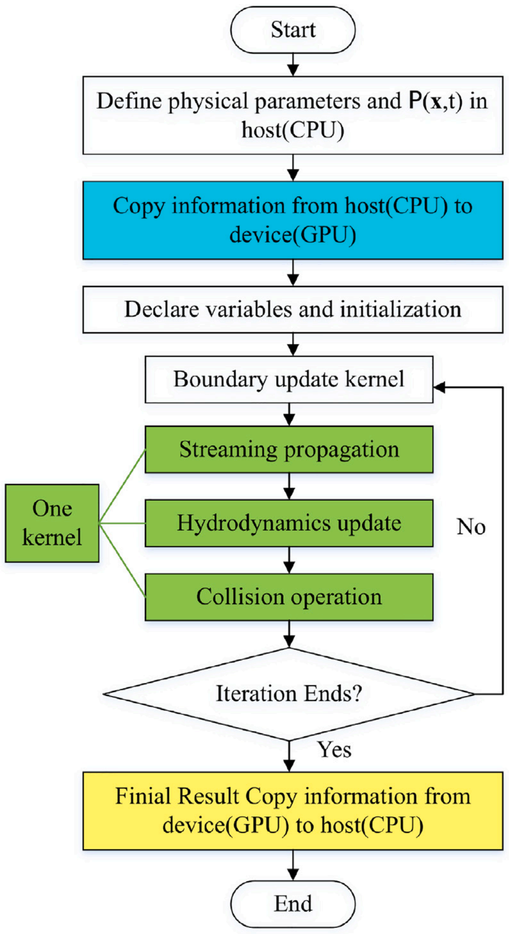
Start
Define physical parameters and P(x,t) in host(CPU)
Copy information from host(CPU) to device(GPU)
Declare variables and initialization
Boundary update kernel
Streaming propagation
Hydrodynamics update
Collision operation
One kernel
No
Iteration Ends?
Yes
Finial Result Copy information from device(GPU) to host(CPU)
End

Fig. 3. Flow chart of GPU Implementation.

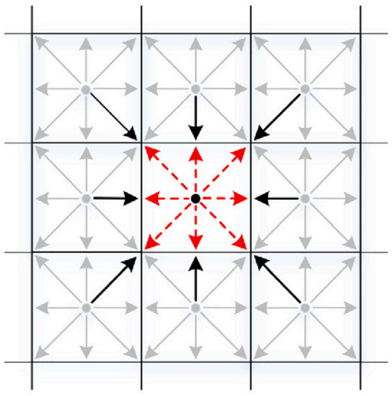

Fig. 4. Propagation scheme: pull scheme with misaligned read and aligned write. The state before streaming and after streaming are represented by the black solid and red dashed arrow line respectively.

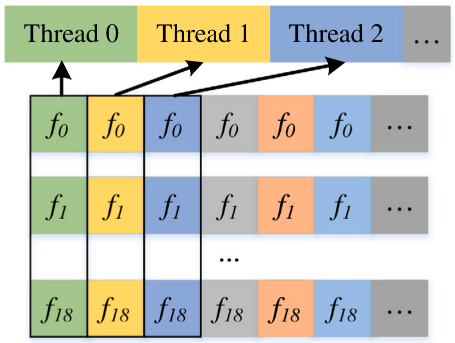

Fig. 5. Data layout: SoA.

2.3. GPU parallelism and optimization

GPU acceleration has been practiced in our research group for a couple of years for different research areas such as turbulence [27], biomedical flows [28,29], and porous media flows [30,31]. The general flow chart of GPU implementation is indicated in Fig. 3. The difference between GPU parallel and CPU serial is the algorithms used for the LBM implementation. To achieve an efficient GPU-LBM algorithm, memory access patterns, register utilization, and overlap of memory transfer and arithmetic operations are the three main aspects to be optimized. For the memory access patterns, an A–B (2 sets of distribution functions) pattern (avoid data dependency), 'pull' scheme (propagation scheme), SoA (data layout), and semi-indirect addressing (memory addressing) are selected in our simulation. Dynamic allocation, register utilization, and multi-GPU are the other critically needed parallel algorithms that can improve the efficiency and capability of the hemodynamic simulation. The detailed description of the parallel algorithms is illustrated as follows.

2.3.1. Dynamic allocation

By default, the compiled model for a code has the characteristic that the instructions and parameters of the code must be linked in the 2 GB static continuous address space. This limitation takes the dominant effect in impeding the simulation of large-scale problems. For example, two set of distribution functions of the lattice Boltzmann model (38 arrays) and flow properties in the biomedical flow, including density, velocity, and pressure(5 arrays), are essential to be defined in the CPU host. From the experience of investigating the hepatic-portal venous system, the mesh size is around $586 \times 532 \times 442$, and the needed memory for our calculations is around 30 GB (for total cells) and 8 GB (for fluid cells). Both of them are much larger than 2 GB. Nowadays, a single NVIDIA GPU card has up to 80 GB of memory for computing. Hence, it is critical to find a way to release this limitation. Dynamic memory (heap) instead of static memory (stack) was applied in our simulations. Dynamic memory (heap) randomly allocates portions from a large pool of memory. However, the transmission array should be logically contiguous when the data copy from the CPU (host) to GPU (device). Hence, in our algorithm, a class of the variable has been developed to manage the allocation and deallocation of memory and utilize the full memory capacity of the GPU cards. The procedures of operation are listed as

- The class can dynamically allocate memory in one dimension array, and each variable only uses a total of $5 \times$ sizeof(float) or sizeof(double) on the stack. Each variable includes three dimensions in different directions, one variable of fluid properties, and data precision. The illustration is like Array3 $<$ precision; 3 dimensions in different directions $>$ a variable of fluid property. Take volumetric parameter $\mathcal{P}$ as an example: the class is Array3 $<$ double; nx; ny; nz $>$ $\mathcal{P}$.
- Assign the memory in an array on the heap in orders in a continuous block through the copy constructor.
- Swapped and destroyed the information, which is related to assignment operators to release the memory.

2.3.2. Propagation scheme

The 'pull' scheme has the opposite propagation direction in contrast to the conventional LBM streaming. The distribution function reads

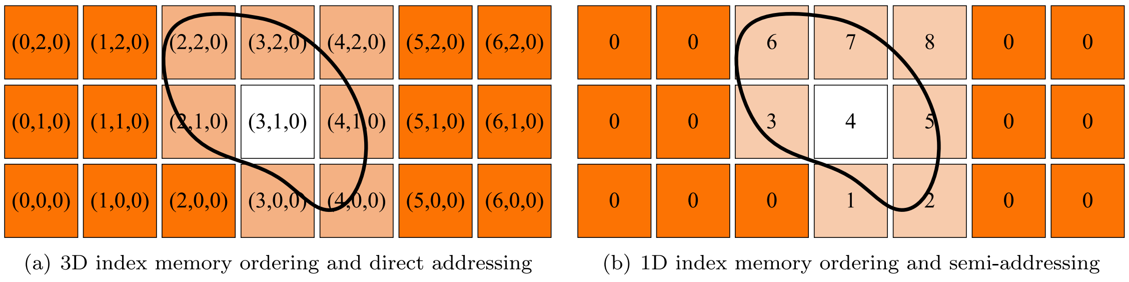

Fig. 6. A computational domain contains fluid, boundary and solid cells with white, light and deep orange, respectively. (a) 3D index memory ordering $a[i][j][k]$ and direct addressing $PositionIndex = i \times ny \times nz + j \times nz + k$, (b) 1D index memory ordering $a[i]$ and semi-addressing $PositionIndex = Positionx[i] \times ny \times nz + Positiony[i] \times nz + Positionz[i]$.

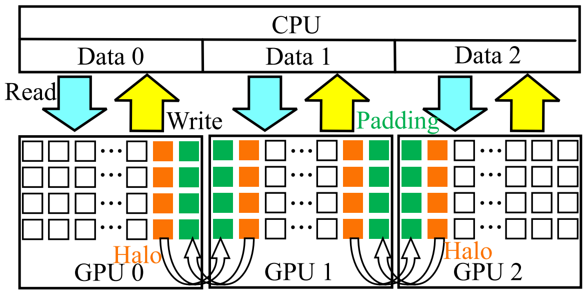

Fig. 7. A schematic of data transfer between GPUs.

from the adjacent cells (black solid arrow lines), and the values propagate to the local cell after streaming (red dashed arrow lines), as Fig. 4 illustrates. Then the data stored in the cell can be used for collision directly. In the implementation, it yields misaligned read and aligned write. Since misaligned read is faster than misaligned write in the GPU [11], the 'pull' scheme is employed in our simulation.

2.3.3. Data layout

Coalesced memory access efficiently reduces the memory latency of a GPU program that results in acceleration. Coalesced memory access means all the threads in a block access the memory address simultaneously. It has been reported that the Structure of Array (SoA) scheme is suitable for GPU acceleration [32]. Hence, multiple-dimension arrays are expanded into one-dimension arrays. We take the D3Q19 model and assume the mesh size $nx \times ny \times nz$ to show the data layout in the GPU. Regularly, the distribution function stores as a 4D array on the CPU, such as $f[x][y][z][i]$. Here x, y, and z represent the space; i ranges from 0 to 18 represent the direction of discrete velocity in the volumetric lattice Boltzmann method. Different structures of one-dimensional arrays in the GPU lead to varying effects on the GPU acceleration. The value of one distribution of all cells in the whole computational domain occupies consecutive elements in memory, which is indicated in Fig. 5 . The distribution functions are addressed as $f[i \times nz \times ny \times nx + x \times ny \times nz + y \times nz + z]$. In this scheme, the number of threads of distribution functions within a wrap (32 threads) can access consecutive memory.

2.3.4. Memory ordering and memory addressing

The geometry employed for analyzing image-based hemodynamics is sparse. And the solid cells in the geometry cost the most for memory storage, but they are excluded from the calculation. From our cases, the fluid volume fraction (q) in the hepatic-portal venous system is $q = 0.2 \backsim 0.3$, i.e., 70% − 80% of the requested memory is wasted. Hence, 1D index memory ordering and semi-addressing are significantly employed to enhance the efficiency of the calculation. As Fig. 6 shows, a computational domain with mesh size $nx \times ny \times nz$ (i.e., 7 × 3 × 1) is taken as an example to indicate the algorithm, which includes one fluid cell (white), seven boundary cells (light orange) and thirteen solid cells (deep orange). The memory cost of traditional 3D index memory ordering $a[i][j][k]$ and direct addressing $PositionIndex = i \times ny \times nz + j \times nz + k$ (i.e., $PositionIndex = i \times 3 + j + k$) (Fig. 6(a)) contains two sets of distribution functions (38 arrays) and flow properties (5 arrays) for each cell, i.e., $\mathbf{nx \times ny \times nz \times (38 + 5)}$. While the requested memory storage for 1D index memory ordering $a[i]$ and semi-addressing $PositionIndex = Positionx[i] \times ny \times nz + Positiony[i] \times nz + Positionz[i]$ (i.e., $PositionIndex = Positionx[i] \times 3 \times 1 + Positiony[i] \times 1 + Positionz[i]$) (Fig. 6(b)) including, two sets of distribution functions (38 arrays) and flow properties (5 arrays) for 8($\Leftrightarrow$ $q \times nx \times ny \times nz$) fluid+boundary cells and one solid cell (can be ignored for the large computational domain), and position information (1 array for PositionIndex of each cell + 3 array for i, j,k of fluid+boundary cells), i.e. $\mathbf{q \times nx \times ny \times nz \times 43 + nx \times ny \times nz + q \times nx \times ny \times nz \times 3}$. Hence, the memory cost for the new one is $46q/43 + 1/43$ times that of the traditional algorithm. That means when q is smaller, 1D index memory ordering and semi-addressing algorithms take more advantage of memory storage.

2.3.5. Register utilization

CUDA provides a memory hierarchy, including device memory (global, constant, and texture memory) and on-chip memory (register, shared memory, and local memory), which has the main differences in latency and memory storage limitation. Global memory is the main part of the device memory, which has a large storage memory and a high latency of 400–800 cycles [33]. The register is an on-chip resource distributed to each thread with nearly zero cycle latency. Hence, register memory is better to be applied in the optimization due to the low latency. The amount of registers is limited, a block contains 65,536 registers for Tesla V100, and the memory for each register is 32-bit. If too many registers are assigned in each thread, it cannot take full utilization in the memory. For example, 43 registers are needed in the D3Q19 per thread and 1024 threads in one block. 43K registers will be used in one block. In this case, the remaining 22K registers are unused, and only one block is applied in the simulation. Then the registers get approximately 60% utilization. The utilization of register memory effectively in each thread is critical in GPU optimization. The selection of the grid and block size is a scheme to get higher register memory utilization which can be predicted by the tool "CUDA_Occupancy_Calculator" provided by CUDA.

2.3.6. Multi-GPU

Multi-GPU parallelism is critically needed in problems that need high resolution. The basic idea of the multi-GPU is splitting the total memory into several parts, which depends on the number of GPU cards and the costing memory, and then reading and writing data with these cards. Before doing the multi-GPU parallel, two properties of the device should be checked. Firstly, can the device do peer-to-peer communication (P2P)? And then, how many GPU cards can be applied in a calculation? As Fig. 7 shows, there are three computational data regions in each GPU card, including internal (white region), halo (orange region), and padding part (green region). First, compute the

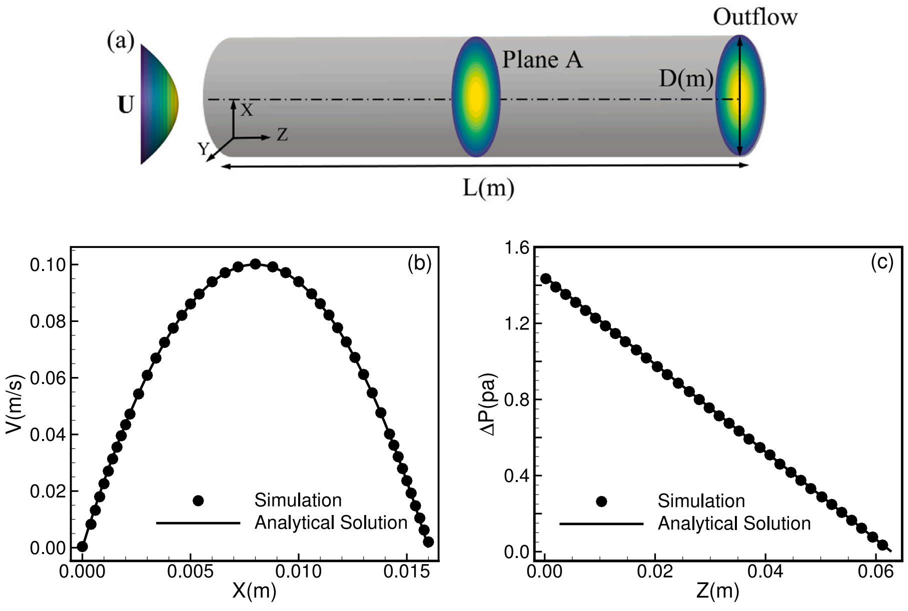

Fig. 8. (a) Schematics for a pipe flow with parabolic velocity inlet. (b) The comparison of the velocity distribution along x-axis direction in center of the Plane A and (c) pressure difference along centerline between simulation and the analytical solution. Black points are recorded from simulation and the black line is generated from the analytical solution.

internal, halo, and padding parts in a GPU device with a stream, then exchange halo data between neighboring GPUs in different streams to update the data of padding. For example, the halo regions in 'GPU 0' will be transferred to the beginning padding part of 'GPU 1', and the padding part in the 'GPU 0' will get the information in the halo regions in the 'GPU 1'. Next, synchronize computation on all devices before proceeding to the next iteration by the function of *cudaDeviceSynchronize()*. Last, the data from all the devices will be written in the CPU part, ignoring the padding part.

Table 1
The relative error between the simulated and analytical velocity and convergence check at point (0.0001,0.0008,0.0032) m when the spatial resolution varies from $16^2 \times 61$ to $128^2 \times 505$ with four levels.

Resolution	$16^2 \times 61$	$64^2 \times 249$	$80^2 \times 313$	$128^2 \times 505$
Vs (m/s)	0.01901	0.02225	0.02321	0.02322
Va (m/s)	0.02344			
$(Vs - Va)/Va$ (%)	18.90	5.05	0.97	0.90
$Convergence$ (%)	14.3	4.1	0.7	/

3. Application studies

In order to present the feasibility of the technology we developed for investigating the hemodynamics in the hepatic-portal venous system, the accuracy and efficiency of the model are presented by three different application cases, including pipe flow, hemodynamics in the portal venous system, and hemodynamics in a simple hepatic-portal venous system. All the application cases are examined with the Intel Broadwell E5-2683 v3@ 2.30 GHz (CPU) and NVIDIA Tesla V100 16 GB (GPU). MFLUPS (million fluid lattice updates per second) and memory cost are key parameters to characterize the computational performance test.

3.1. Pipe flow

A blood flow in a pipe with length (L) 0.064 m and diameter (D) 0.016 m is selected in the range of length and diameter of the portal vein, and the geometry information is directly from the image. The density and kinematic viscosity of blood is $\rho = 1060\ \mathrm{kg/m^3}$ and $\nu = 3.3 \times 10^{-6}\ \mathrm{m/s^2}$, respectively. As shown in Fig. 8(a), the blood is driven by the velocity with a parabolic profile in the z-axis direction and the maximum velocity(U_{zmax}) is set as 0.1 m/s which is dependent on the blood flow rate of the portal vein. In this case, the outlet boundary is fully developed boundary condition. For such a flow, the analytical solution of the velocity profile distributed in the XY-plane is $U_z = U_{zmax}(1 - 4((x - R)^2 + (y - R)^2)/D^2)$, and the pressure difference from an outlet along the z-axis direction is $\triangle\ P = 16\rho\nu U_{zmax}z/(gD^2)$. The spatial resolution selection is the primary step for simulation which is dependent on the convergence check of velocity and relative error between the simulated (Vs) and the analytical velocity (Va). The point $(0.001, 0.008, 0.032)$ m in Plane A is selected as the test point due to the point close to the wall has a sensitive connection with the resolution. As Table 1 listed, 80 × 80 × 313 is chosen to conduct the test with consideration of computational cost and accuracy. Based on the appropriate resolution, we select Plane A and the centerline of the pipe to record the velocity profile along the x-axis direction with $y = 0.008$ m and pressure difference along the z-axis direction, respectively. As Fig. 8(b) and (c) indicated, both velocity distribution and pressure differences are in good agreement with the analytical solution. Hence, the test case confirms that the technology we developed is reasonable for the analysis of image-based computational fluid dynamics.

From our experience in GPU acceleration, the speed-up of GPU algorithm for regular dense geometry and CPU-serial is around 4000 with the Intel Broadwell E5-2683 v3@ 2.30 GHz (CPU) and NVIDIA Tesla V100 16 GB (GPU). Here, we focus on the speed-up and memory saving ratio between the GPU algorithm of regular dense geometry (DENSE) and sparse geometry (SPARSE). The main differences between DENSE and SPARSE are in memory ordering and addressing. The performance of GPU acceleration of pipe with fluid volume fraction $q = 0.766$ is indicated in Table 2 for three different resolutions through speed-up and memory saving ratio. The speed-up of different resolutions is nearly the same. SPARSE is faster when the mesh size is smaller. And the memory saving ratio between DENSE and SPARSE is 0.85 which is related to the fluid volume fraction. Hence, the SPARSE has a similar performance to DENSE when the fluid volume fraction is close to 1.

3.2. Hemodynamics in the portal venous system

In this section, the accuracy and efficiency of the technology we developed for image-based sparse geometry are presented by analyzing hemodynamics in the portal vein system. Here, we take one anonymous

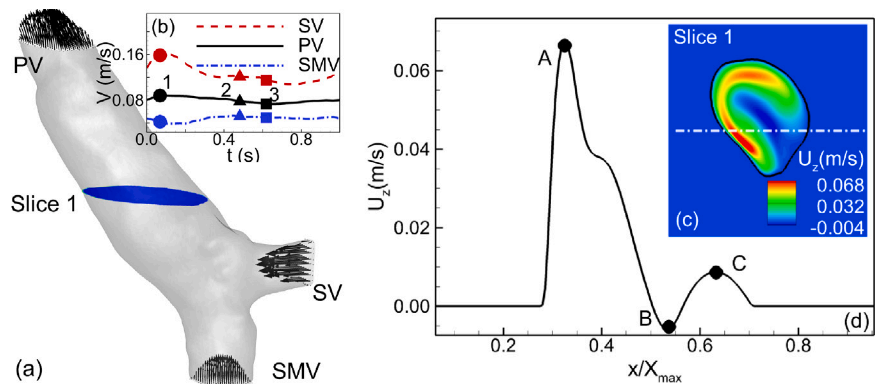

Fig. 9. (a) Schematics for the portal venous system with parabolic velocity inlet (SMV and SV) and outlet (PV). Slice 1 is a representative plane extracted to analyze the characteristics of the flow field. (b) The pulsatile flow rate of PV (solid black line), SMV (dash–dot blue line), and SV (dash red line) obtained from the MRI images, and three representative time point 1(circle), 2(triangle), and 3(rectangle) which has the maximum, mean, and minimum velocity of PV are selected to study the effect of pulsatile. (c) Velocity distribution of Slice 1, the dash–dot white line is the centerline of Slice 1 in the y-direction. (d) The velocity profile of the centerline of Slice 1, A, B, and C are three representative points for the convergence check.

Table 2
The GPU performance of pipe flow with $q = 0.766$ for SPARSE comparing with DENSE with speed-up and memory saving ratio through three different resolutions.

Resolution	Speed (MFLUPS)		Speed-up	Memory cost (GB)		Saving ratio
	DENSE	SPARSE		DENSE	SPARSE	
64 × 64 × 249	1937	2172	1.12	0.23	0.19	0.83
128 × 128 × 507	1966	1975	1.00	1.85	1.57	0.85
184 × 184 × 731	1993	1810	0.91	5.52	4.70	0.85

Table 3
The convergence check at point A, B, and C when the spatial resolution varies from 73 × 80 × 96 to 184 × 200 × 242 with four levels.

Resolution	73 × 80 × 96	103 × 112 × 135	132 × 143 × 174	184 × 200 × 242
V_{zA} (m/s)	0.0780	0.0723	0.0684	0.0678
$Convergence_A$ (%)	7.88	5.77	0.84	/
V_{zB} (m/s)	−0.0098	−0.0063	−0.0059	−0.0058
$Convergence_B$ (%)	36.03	6.06	0.96	/
V_{zC} (m/s)	0.0100	0.0081	0.0086	0.0087
$Convergence_C$ (%)	18.47	6.40	0.83	/

patient with portal hypertension cases as an example. The model of the portal venous system (Fig. 9(a)) is segmented from the patient's MRI medical images, including the superior mesenteric vein (SMV), splenic vein (SV), and portal vein (PV). And blood flow rate of the SMV, SV, and PV in a cardiac cycle (Fig. 9(b)) is recorded from the corresponding MRI images. Three representative time points, 1, 2, and 3, are selected to analyze the hemodynamics in the portal venous system. The point 1 (circle), 2 (triangle), and 3 (rectangle) are the time points $t = 0.07, 0.48, 0.62$ s, which have the maximum, mean, and minimum velocity of PV, respectively. Slice 1, located in the middle of the z-direction of the portal venous system, is a representative plane extracted to analyze the characteristics of the flow field. The velocity distribution of Slice 1 is indicated in Fig. 9(c). Three representative points, A, B, and C, in the velocity profile (Fig. 9(d)) along the centerline of the Slice 1 are selected as the tracing points to get the appropriate resolution for simulation. Quantitatively, the convergence check is shown in Table 3 with the specific point A, B, and C. With the suitable resolution 132 × 143 × 174, the 3D flow fields of the three representative times are shown in Fig. 10(a), (b), and (c) stands for the different time points. As time varies, the velocity magnitude varies, but the velocity distribution is similar.

Hence, a hypothesis that the blood flow rate of the portal venous system can be treated as steady flow instead of pulsatile flow is generated. The boundary condition of the portal venous system is changed as a constant velocity to verify this hypothesis. And the constant velocity boundary is selected from three representative time points of PV and SMV, which can be treated as three specific steady cases. Due to mass conservation, the velocity of SV is dependent on the flow rate of PV and SMV. Slice 1 extracted from 3D geometry, as Fig. 9(a) showing, is selected as the representative plane. It can be observed that the velocity distribution is similar in the large velocity magnitude and slight difference when the velocity magnitude is close to 0, comparing the velocity distribution of Slice 1 of steady flow (Fig. 11(a)) with the pulsatile flow (Fig. 11(b)) at the same time. Therefore, the velocity of a vein can be treated as a steady flow consistent with the general knowledge of medicine.

Meanwhile, the pressure difference between PV and SV is captured as $3.43, 5.51, 7.71$ Pa at time points 1, 2, and 3, respectively. Compared with the normal pressure difference of the hepatic-portal venous system 5 mmgh, the pressure difference in the portal venous system can be neglected, which is consistent with clinical treatment. As a result, the model can be considered reasonable with qualitative observation.

The performance of GPU acceleration of the portal venous system is indicated in Table 4. Here, we only take three anonymous patient cases as the sample. The fluid volume fraction q for case1–3 is 0.131, 0.192, and 0.242, respectively. The speed-up is up to 2.2 between SPARSE and DENSE. And the memory saving ratio between DENSE and SPARSE is $0.15 - 0.28$, which depends on q.

3.3. *Hemodynamics in hepatic-portal venous system*

Based on the GPU performance tested by the portal venous system, we take the appropriate resolution of the portal venous system to build a portal venous system with a mesh size of 586 × 532 × 442. As Fig. 12(a) shows portal venous system consists of three hepatic veins (HV) extending from the inferior vena cava (IVC), liver, and portal

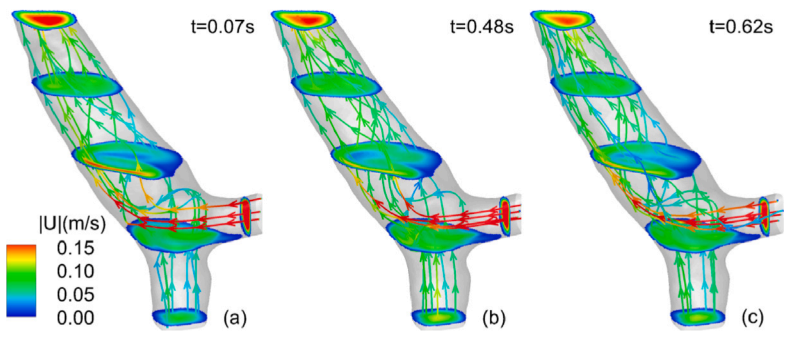

Fig. 10. The 3D flow fields of the three representative time $t = 0.07, 0.48, 0.62$ s for (a), (b), and (c) respectively.

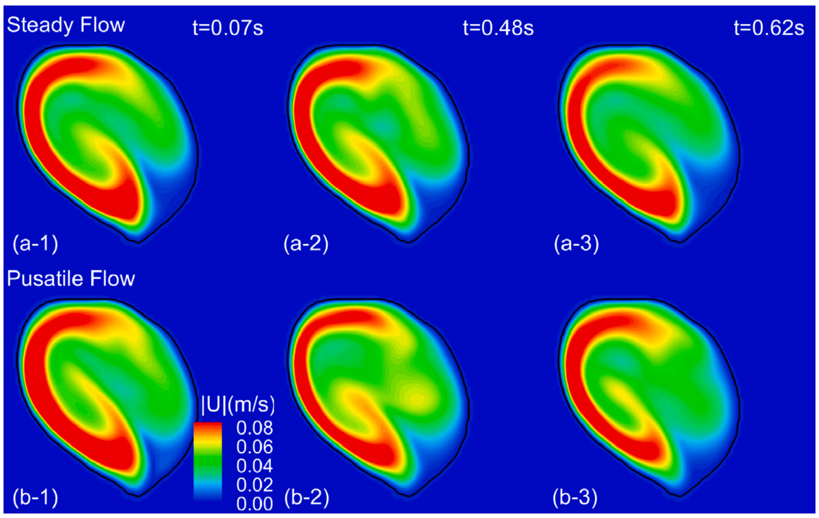

Fig. 11. Comparing (a) Velocity magnitude distribution in Slice 1 of three specific steady cases: (a-1)~(a-3) constant velocity boundary condition of time points 1~3 of PV and SMV with (b-1)~(b-3) Velocity magnitude distribution in Slice 1 of pulsatile flow at the corresponding time $t = 0.07, 0.48, 0.62$ s.

Table 4
The GPU performance of portal venous system for SPARSE comparing with DENSE with speed-up and memory saving ratio through three different patient cases.

Case	Resolution	Speed (MFLUPS)		Speed-up	Memory cost (GB)		Saving ratio
		DENSE	SPARSE		DENSE	SPARSE	
1 ($q = 0.131$)	132 × 143 × 174	676	1487	2.20	0.73	0.11	0.15
	184 × 200 × 242	779	1532	1.97	1.99	0.31	0.15
	224 × 243 × 294	822	1513	1.84	3.58	0.55	0.15
2 ($q = 0.192$)	184 × 145 × 184	831	1245	1.50	1.10	0.24	0.22
	224 × 176 × 224	863	1230	1.43	1.97	0.44	0.22
	264 × 208 × 263	952	1651	1.73	3.23	0.72	0.22
3 ($q = 0.242$)	184 × 173 × 159	890	1937	2.18	1.13	0.31	0.28
	224 × 210 × 193	914	1659	1.82	2.03	0.56	0.28
	268 × 252 × 231	945	1778	1.88	3.49	0.97	0.28

venous system. The liver can be modeled as porous media to study portal hypertension quantitatively which will be discussed in a future paper. Here we treat the liver as a chamber, i.e., the porosity is 1 with full of blood flow, to see the possibility of simulation with existing computer hardware. The velocity magnitude distribution of the blood flow in the hepatic-portal venous system is shown in Fig. 12(b). The direction of the flow and velocity in the hepatic vein is much larger than others is reasonable. With DENSE, the memory cost is 30.80 GB. It cannot be done on a single GPU 16 GB. Here, we apply Multi-GPU for DENSE. Table 5 separately exhibits the GPU acceleration by two, three, or four GPU cards. The performance accelerates when the number of GPU cards increases. With SPARSE, the memory cost is 9.25 GB, the memory saving ratio is 0.3, and the speed is 1378 MLUPS. In other words, a numerical case can be done around half an hour with parallel execution of SPARSE, whereas it takes 83 days with serial execution from our previous experience.

4. Summary & future work

To develop a new technique monitoring portal hypertension by PPG based on non-invasive measurement (MRI images) instead of current invasive measurement, massive parametrization and classification are required. Meanwhile, the hepatic-portal venous system used for simulation consists of HV extending from IVC, liver, and portal venous system, which has a large computational domain. Hence, combining the features of GPU with VLBM is significant in achieving an efficient

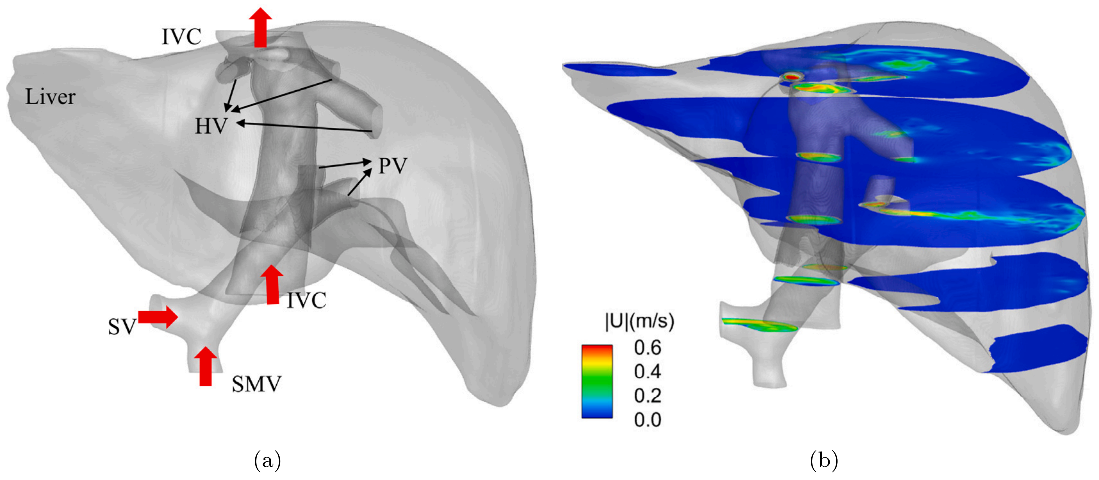

Fig. 12. (a) Schematics for hepatic-portal venous system with parabolic velocity inlet (SMV, SV, IVC before HV) and outlet (IVC after HV). Two PVs and three HVs are connected with liver. (b) Velocity magnitude distribution of the blood flow in hepatic-portal venous system.

Table 5
The GPU performance of hepatic-portal venous system with $q = 0.266$ for SPARSE comparing with multi-GPU DENSE with speed-up through different numbers of GPU card.

Resolution	Speed (MFLUPS)				Memory cost (GB)	
	DENSE (GPU no.)			SPARSE	DENSE	SPARSE
	2	3	4			
$586 \times 532 \times 442$ ($q = 0.266$)	2037	2824	3751	1378	30.80	9.25

GPU-VLBM algorithm. In this paper, GPU parallelization and optimization for the volumetric lattice Boltzmann model are performed, including an A–B (2 sets of distribution functions) pattern, dynamic allocation, SoA, 'push' propagation scheme, semi-indirect addressing, register utilization, and multi-GPU. Three different application cases, including pipe flow, hemodynamics in the portal venous system, and hemodynamics in the hepatic-portal venous system, are employed to prove the feasibility of the method based on accuracy and efficiency. From our previous experience [27,30] in the GPU acceleration, the speed-up of DENSE compared with CPU-serial is around 4000 by the Intel Broadwell E5-2683 v3@ 2.30 GHz (CPU) and NVIDIA TeslaV100 16 GB (GPU). In this paper, we focus on the speed-up and memory saving ratio between DENSE and SPARSE. The results derived from the paper are summarized as follows:

- The reliability of the model is validated qualitatively by the analytical solution of velocity and pressure difference distribution of pipe flow and quantitatively by the portal venous system so that the pulsatility of velocity and pressure difference can be neglected.
- The performance of the application cases shows that the SPARSE presented in the paper has a similar speed with DENSE when the fluid volume fraction (q) is close to 1. While SPARSE speed up to 2.2 times compared with DENSE when q is in the range of 0.19~0.27. Correspondingly, the memory cost saving ratio is dependent on q. In this condition, a numerical case in the hepatic-portal venous system can be converged for around half an hour with parallel execution of SPARSE. The time for simulation will be decreased by the number of GPU cards increasing with the multi-GPU implementation.

In summary, the method proposed in the paper to investigate a new technique to monitor portal hypertension in the hepatic-portal venous system is the feasibility. While the developing liver model is still a huge challenge. We still work on the liver model based on the fractal theory. This new technology is believed to improve patient care and clinical decision-making in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research is supported by National Natural Science Foundation of China (Nos. 11872062, 12102419), Fundamental Research Funds for the Provincial Universities of Zhejiang No. 2020YW05. And this work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation, United States grant number, ACI-1053575 [34]. And thanks to Dr. QiDong Wang from First Affiliated Hospital, College of Medicine, Zhejiang University, who provided all the MRI images.

References

[1] Mauro E, Gadano A. What's new in portal hypertension? Liver Int 2020;40:122–7.
[2] Simonetto DA, Liu M, Kamath PS. Portal hypertension and related complications: diagnosis and management. In: Mayo clinic proceedings. 94, (4):Elsevier; 2019, p. 714–26.
[3] Berzigotti A, Seijo S, Reverter E, Bosch J. Assessing portal hypertension in liver diseases. Expert Rev Gastroenterol Hepatol 2013;7(2):141–55.
[4] De Franchis R. Expanding consensus in portal hypertension: Report of the baveno vi consensus workshop: Stratifying risk and individualizing care for portal hypertension. J Hepatol 2015;63(3):743–52.
[5] Bosch J, Abraldes JG, Groszmann R. Current management of portal hypertension. J Hepatol 2003;38:54–68.
[6] Qi X, An W, Liu F, Qi R, Wang L, Liu Y, Liu C, Xiang Y, Hui J, Liu Z, et al. Virtual hepatic venous pressure gradient with CT angiography (CHESS 1601): a prospective multicenter study for the noninvasive diagnosis of portal hypertension. Radiology 2019;290(2):370–7.
[7] Wittmann M, Zeiser T, Hager G, Wellein G. Comparison of different propagation steps for lattice Boltzmann methods. Comput Math Appl 2013;65(6):924–35.

[8] Tölke J, Krafczyk M. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. Int J Comput Fluid Dyn 2008;22(7):443–56.
[9] Tölke J. Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA. Comput Vis Sci 2010;13(1):29–39.
[10] Bailey P, Myre J, Walsh S, Lilja D, Saar M. Accelerating lattice Boltzmann fluid flow simulations using graphics processors. In: Parallel processing, 2009. ICPP'09. International conference on. IEEE; 2009, p. 550–7.
[11] Obrecht C, Kuznik F, Tourancheau B, Roux J. A new approach to the lattice Boltzmann method for graphics processing units. Comput Math Appl 2011;61(12):3628–38.
[12] Ryoo S, Rodrigues C, Baghsorkhi S, Stone S, Kirk D, Hwu W. 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.
[13] Tran N, Lee M, Hong S. Performance optimization of 3D lattice Boltzmann flow solver on a GPU. Sci Program 2017;2017.
[14] Herschlag G, Lee S, Vetter JS, Randles A. GPU data access on complex geometries for D3Q19 lattice Boltzmann method. In: 2018 IEEE international parallel and distributed processing symposium. IPDPS, IEEE; 2018, p. 825–34.
[15] Herschlag G, Lee S, Vetter JS, Randles A. Analysis of GPU data access patterns on complex geometries for the D3Q19 lattice Boltzmann algorithm. IEEE Trans Parallel Distrib Syst 2021;32(10):2400–14.
[16] NVIDIA Corporation. CUDA Toolkit Documentation. 2015, Accessed: September 2015, http://docs.nvidia.com/cuda/index.html.
[17] Tomczak T, Szafran RG. A new GPU implementation for lattice-Boltzmann simulations on sparse geometries. Comput Phys Comm 2019;235:258–78.
[18] Huang C, Shi B, Guo Z, Chai Z. Multi-GPU based lattice Boltzmann method for hemodynamic simulation in patient-specific cerebral aneurysm. Commun Comput Phys 2015;17(4):960–74.
[19] Shealy BT, Yousefi M, Srinath AT, Smith MC, Schiller UD. GPU acceleration of the hemelb code for lattice boltzmann simulations in sparse complex geometries. IEEE Access 2021;9:61224–36.
[20] Xian W, Takayuki A. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. Parallel Comput 2011;37(9):521–35.
[21] Blair S, Albing C, Grund A, Jocksch A. Accelerating an MPI lattice Boltzmann code using OpenACC. In: Proceedings of the second workshop on accelerator programming using directives. ACM; 2015, p. 3.
[22] Yu H, Chen X, Wang Z, Deep D, Lima E, Zhao Y, Teague D. Mass-conserved volumetric lattice Boltzmann method for complex flows with willfully moving boundaries. Phys Rev E 2014;89(6):063304.
[23] Zhang X, Gomez-Paz J, Chen X, McDonough JM, Islam MM, Andreopoulos Y, Zhu L, Yu H. Volumetric lattice Boltzmann method for wall stresses of image-based pulsatile flows. Sci Rep 2022;12(1):1–15.
[24] Yu H, Khan M, Wu H, Zhang C, Du X, Chen R, Fang X, Long J, Sawchuk AP. Inlet and outlet boundary conditions and uncertainty quantification in volumetric lattice Boltzmann method for image-based computational hemodynamics. Fluids 2022;7(1):30.
[25] Yu H, Khan M, Wu H, Du X, Chen R, Rollins DM, Fang X, Long J, Xu C, Sawchuk AP. A new noninvasive and patient-specific hemodynamic index for the severity of renal stenosis and outcome of interventional treatment. International Journal for Numerical Methods in Biomedical Engineering 2022;38(7):e3611.
[26] Yu H. Non-invasive functional assessment technique for determining hemodynamic severity of an arterial stenosis 2022, Google Patents, US Patent 11, 538, 153.
[27] Yu HW, Chen R, Wang H, Yuan Z, Zhao Y, An Y, Xu Y, Zhu L. GPU accelerated lattice Boltzmann simulation for rotational turbulence. Comput Math Appl 2014;67(2):445–51.
[28] Wang Z, Chen N, Zhao Y, Yu H. GPU-accelerated lattice Boltzmann method for extracting real biomechanical geometry and volumetric boundary condition. Comput & Fluids 2015;115:192–200.
[29] Cui J, Wu T, Liu Y, Fu BM, Jin Y, Zhu Z. A three-dimensional simulation of the dynamics of primary cilia in an oscillating flow. Appl Math Model 2022;108:825–39.
[30] An S, Yu H, Wang Z, Chen R, Kapadia B, Yao J. Unified mesoscopic modeling and GPU-accelerated computational method for image-based pore-scale porous media flows. Int J Heat Mass Trans 2017;115:1192–202.
[31] An S, Yu H, Yao J. GPU-accelerated volumetric lattice Boltzmann method for porous media flow. J Petro Sci Eng 2017;156:546–52.
[32] Delbosc N, Summers J, Khan A, Kapur N, Noakes C. Optimized implementation of the lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation. Comput Math Appl 2014;67(2):462–75.
[33] Power J, Hill M, Wood D. Supporting x86-64 address translation for 100s of GPU lanes. In: 2014 IEEE 20th international symposium on high performance computer architecture. HPCA, IEEE; 2014, p. 568–78.
[34] Towns J, Cockerill T, Dahan M, Foster I, Gaither K, Grimshaw A, Hazlewood V, Lathrop S, Lifka D, Peterson G, et al. XSEDE: accelerating scientific discovery. Comput Sci Eng 2014;16(5):62–74.