

Physically based visual simulation of the Lattice Boltzmann method on the GPU: a survey

Octavio Navarro-Hinojosa¹  · Sergio Ruiz-Loza¹ ·
Moisés Alencastre-Miranda¹

Published online: 27 April 2018

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract The rapid increase in performance, programmability, and availability of graphics processing units (GPUs) has made them a compelling platform for computationally demanding tasks in a wide variety of application domains. One of these is real-time computational fluid dynamics, which are computationally expensive due to a large number of grid points that require calculations. One commonly used tool to simulate fluid flows is the Lattice Boltzmann method (LBM), mainly due to its simpler formulation when compared to solving the Navier–Stokes equations, and because of its scalability on parallel processing systems. In this paper, we give an up-to-date survey on the research regarding the LBM for fluid simulation using GPUs. We discuss how the method was implemented with different GPU architectures and software frameworks, focusing on optimization techniques and their performance. Additionally, we mention some applications of the method in different areas of study.

Keywords LBM · GPU · CFD · CUDA · OpenCL · OpenACC

1 Introduction

Since Evans et al. [23] began publishing on the Particle-in-Cell (PIC) method for hydrodynamic calculations in 1957, computational fluid dynamics (CFD) has grown

✉ Octavio Navarro-Hinojosa
a00967953@itesm.mx; hagennav@gmail.com

Sergio Ruiz-Loza
sergio.ruiz.loza@itesm.mx

Moisés Alencastre-Miranda
malencastre@itesm.mx

¹ IT and Computer Department, Tecnológico de Monterrey, 01389 Santa Fe, Mexico City, Mexico

as a research field for both numerical methods and algorithms (Holt [39], Ferziger and Peric [27] have reviewed advances in the former, while Tan and Yang [77] presented a survey on the latter), and although parallel CFD numerical methods had already been proposed by 2004—for example [1,22] for distributed memory multiprocessor computers—it is in 2003, with the emergence of programmable commodity graphics hardware, that Goodnight et al. [30] showed that finite-difference methods for solving partial differential equations (PDE) can be mapped onto graphics processors, using the pixel pipeline and the fast on-card video memory. Also in 2003, Li et al. [51] accelerated the computation of the Lattice Boltzmann method (LBM) on general-purpose graphics processors, by grouping particle packets into 2D textures, and mapping the Boltzmann equations to the rasterization and frame buffers.

In 2004 Harris [34] introduced a technique for fast fluid dynamics simulations, showing that the “stable fluids” method by Stam [76] could be implemented on the GPU with the specialized Cg language, obtaining a $6 \times$ speedup over an equivalent CPU simulation. Then in 2007, Brandvik and Pullan [11,12] reported a $40 \times$ speedup using a GPU compared to the CPU implementation of a 2D Euler solver, and a $16 \times$ speedup for the 3D solver; the authors implemented their solution with massive parallelism controlled by the CUDA [20] interface, concluding that “this type of many-core architecture, be it on the GPU or future incarnations of the CPU, is likely to be the future of scientific computation”. Parallel implementation of CFD algorithms has received great interest indeed, and today hybrid CPU–GPU heterogeneous computing [97], where the GPU serves as a co-processor to the multi-core CPU (considered by Posey [68] and Gaudlitz et al. [28]) is being explored for high-performance computing (HPC) environments.

Although there are several CFD algorithms to simulate physically based fluid animation using the GPU (Niemeyer and Sung [59] reviewed the progress made in developing GPU-based CFD solvers, focusing on incompressible NS equations, laminar, turbulent, and reactive flows), one that has become popular recently and is used in a variety of applications is the LBM. Due to its formulation and dynamics, the LBM has some advantages over other physically based fluid animation methods, especially in algorithm parallelization [44]. Given the relevance of parallel computing applied to physically based fluid animation using the LBM, in this work we survey articles focusing on GPU implementations and their applications, motivated by the objectives of identifying optimization techniques, and obtaining a performance reference.

This paper is organized as follows. Section 2 gives a brief overview of the basic LBM algorithm, and the extensions needed to simulate free surface flows. Additionally, a brief description of the software frameworks used to program the GPU is given. In Sect. 3 we examine various approaches to simulate LBM flows implemented on GPUs. Section 4 showcases several applications of the LBM where GPU implementations resulted in increased performance when compared to their CPU versions. Section 5 discusses further challenges and potential research directions. Finally, Sect. 6 concludes with a discussion of techniques that reported performance gains, as well as the challenges of implementing the LBM using GPUs.

2 Background

2.1 Lattice Boltzmann method overview

The LBM is a rather new approach to approximate the NS equations. The basic idea is to construct models that incorporate the microscopic and mesoscopic physical processes so that the macroscopic properties obey the desired macroscopic equations (the Lattice Boltzmann equation converges to the Navier–Stokes equation) [44].

The LBM can be described as a Cellular Automata representing discrete packets moving on a discrete regular lattice at discrete time steps. At each grid cell, there are variables indicating the status of that grid point; each one is modified at each time step based on linear and local rules. Each grid cell stores a set of distribution functions (DFs), which represent an amount of fluid moving with a fixed velocity.

The Lattice Boltzmann equation is the governing equation of the LBM:

$$f_i(x + e_i \Delta x, \Delta t) = f_i(x, t) + \omega_i(f(x, t)) \tag{1}$$

where f_i is the velocity distribution function in the i th direction e_i , and $\omega_i(f(x, t))$ is the collision operator representing the rate of change of f_i resulted from the collision. The Lattice Boltzmann equation allows particles to move only in a set of discrete velocities. Usual sets are 9 velocities in two dimensions, and 13, 15, 19, or 27 velocities in three dimensions. The naming convention of Lattice Boltzmann velocity, set by Qian et al. [70], is $DdQq$, with d being the number of dimensions and q being the number of discrete velocities, respectively; some of these are given in Fig. 1. These are also referred to as stencils.

The basic LB algorithm consists of two steps, the stream (or advection) step, and the collision step. In the first step, all velocity distribution functions are convected with their respective velocities. This propagation results in a movement of the particle distributions from a given cell to the neighboring cells. Formulated in terms of distribution functions, it can be written as:

$$f_i^*(x, t + \Delta t) = f_i(x - \Delta t e_i, t) \quad i = 0, 1, \dots, M \tag{2}$$

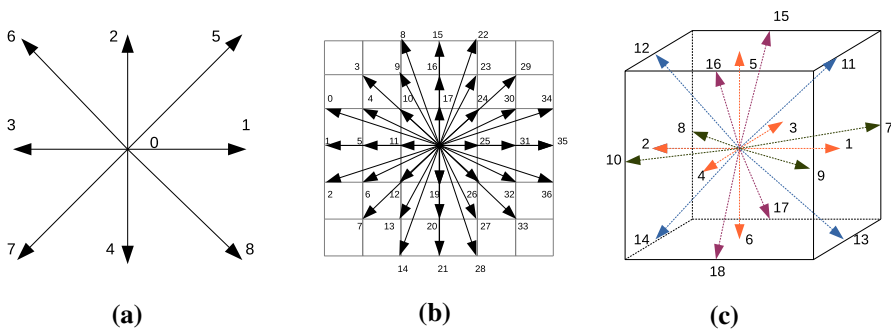


Fig. 1 Stencils used for the LBM. a D2Q9, b D2Q37, c D3Q19

The collision step of the LBM allows the simulation of incompressible fluids by considering the collision between particles. This is done by weighing the DFs of a cell with the *equilibrium distribution functions*, which depend solely on the density and velocity of the fluid, and are denoted by f_i^{eq} . Density and velocity can be computed by adding all the DFs for one cell:

$$\rho = \sum f_i \quad u = \sum e_i f_i \quad (3)$$

The equilibrium distribution function can be computed with:

$$f_i^{\text{eq}} = w_i \left[\rho + 3e_i \cdot u - \frac{3}{2}u^2 + \frac{9}{2}(e_i \cdot u)^2 \right] \quad (4)$$

where w_i constants depend on the employed lattice geometry.

The equilibrium DFs represent a stationary state of the fluid. Collisions of the molecules in a real fluid are approximated by linearly relaxing the DFs of a cell toward their equilibrium state. Thus, each f_i is weighed with the corresponding f_i^{eq} using:

$$f_i(x, t + \Delta t) = (1 - \omega) f_i^*(x, t + \Delta t) + \omega f_i^{\text{eq}}. \quad (5)$$

Here, $\omega \in (0 \dots 2]$, is the parameter that controls the viscosity of the fluid; values close to 0 result in very viscous fluids, while values near 2 result in more turbulent flows. Often, $\tau = 1/\omega$ is also used to denote the lattice viscosity. More details of the algorithm are given in [36,80].

Performance of Lattice Boltzmann simulations is measured in lattice updates per second (LUPS), which indicates the number of lattice collision and streaming steps performed in one second. For recent implementations, a more convenient unit of measurement is million lattice node updates per second (MLUPS) [4]. The metric million fluid lattice cell updates per second (MFLUPS) can also be used, and represents a value where only the fluid nodes are counted [32,84]. The MLUPS are computed as:

$$\text{Performance} = \frac{nx \times ny \times nz \times \text{loopnumber} \times 10^{-6}}{\text{totaltime}} \text{MLUPS}. \quad (6)$$

2.1.1 Free surface flow with LBM

For the purposes of simulating fluids, the most important difference between gases and liquids is that a liquid has a free surface. A free surface is the boundary between two fluids, commonly perceived by the human eye as a boundary between a liquid and a gas such as water and air [19,80,81]. As with all matter, when a fluid moves, it has to conserve mass: without any external influence, a fluid in a closed system should maintain the same mass from one time step to the next. This is of vital importance for simulation accuracy, especially for simulations with free surfaces. Because of this, interactions of a fluid at its free surface must be managed to ensure that fluid exchanges remain balanced, that the mass of the fluid remains constant, and that the free surface moves correctly through the simulation grid.

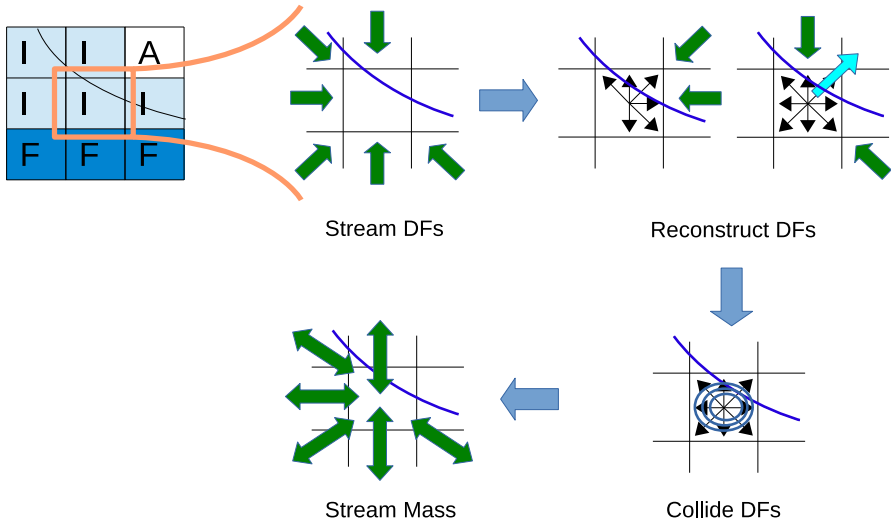


Fig. 2 Steps that have to be executed for interface cells

Level sets have been successfully used to model free surfaces [67,81]. The basic idea is to define a continuous function that represents the fluid surface and then to advect that function according to local fluid velocities. This approach has mathematical advantages that make it relatively simple to model behaviors that have the potential to add significant complexity. The disadvantage of using level sets is that they are not effective at conserving mass [80].

The volume of fluid (VOF) method uses a simple scheme of tracking a fractional value to represent the percentage of each cell at the fluid’s surface [38,80,82]. Unlike level sets, which represent a continuous boundary for the fluid surface, the VOF method uses the simulation grid to form a discretized fluid surface boundary. A significant benefit of this method is that, by aligning the fluid surface with the simulation grid, the VOF method implies a smaller memory footprint than other methods [38]. This is essential when working with the reduced memory available on GPUs.

To aid in tracking of free surfaces, two additional cell types were introduced: interface cells, and empty cells. Empty cells have no fluid, while partially filled interface cells separate empty cells from fluid cells. Cell type conversions, directed by boundary conditions, manage when interface cells become filled or emptied. Empty cells that contain no fluid do not need to be considered in the algorithm until they are converted to interface cells. Figure 2 shows additional steps to be considered for the simulation of interface cells.

3 Lattice Boltzmann GPU implementations

Even though the method is faster than other solutions to the NS equations, the computation of the LBM is still slow: several equations and data transfers have to be performed

for each cell of the grid. However, the following characteristics of the LBM allow it to be parallelized with GPUs [41,44,51,80]:

- LBM works on a Cartesian grid, with each cell functioning independently.
- Calculations at each cell are simple, but there are usually a large number of cells.
- Transfer of data between the cell centers is ordered and can be utilized to make data access patterns conducive for implementing on a GPU.

There are many works that have taken advantage of these characteristics and have parallelized the LBM using GPUs. In the following sections, we have focused on works that parallelize the basic algorithm and its free surface extension, since these form the base for other applications. Additionally, we also focus our attention on multi-GPU implementations. Based on the methodology used, we classify these approaches into two categories: using textures and render buffers to perform computations, and using GPGPU APIs.

3.1 Using textures and render buffers to perform computations

The first implementations of LBM on graphics hardware were achieved by solving general equations to rendering operations; this approach was the only option researchers had because there were no dedicated software APIs or frameworks that enabled them to use graphics processing units for general-purpose computing. Researchers had to come up with specific techniques, or had to modify certain parts of the algorithms, in order to take advantage of the GPU's pipeline and its parallel capabilities. Table 1 shows a summary of LBM implementations results using textures, ordered chronologically. Although MLUPS is a better performance indicator, speedups are presented here since the authors used this metric to show their results. Additionally, since researchers often tested their approaches on different sized lattices, only the best performing ones are presented.

Table 1 Performance obtained for the LBM using texture mapping on the GPU

References	GPU	Grid size	Stencil	Speedup times
Li et al. [51]	GeForce 4 Ti 4600	256^3	D3Q19	55.9
Wei et al. [91]	GeForce 4 Ti 4600	$51 \times 27 \times 27$	D3Q19	9.2
Qiu et al. [71]	GeForce FX 5950 Ultra	$90 \times 30 \times 60$	D3Q13	8.02
Fan et al. [24]	GeForce FX 5800 Ultra	$480 \times 400 \times 80$	D3Q19	4.6
Li et al. [50]	GeForce FX 5900 Ultra	128^3	D3Q19	15
Zhu et al. [103]	GeForce 6600 GT	32^3	D3Q19	3
Zhao et al. [102]	GeForce 6800 Ultra	64^3	D3Q13	13.5
Zhao et al. [100]	GeForce 6800 Ultra	50^3	D3Q13	7.1
Zhao et al. [101]	Quadro FX 4500	41^3 coarse, 35×33^2 fine	D3Q19	28

The speedup reported corresponds to the LBM implementation only

Some of the first researchers to parallelize the LBM using GPUs were Li et al. [51] in 2003, who accelerated the computation of the LBM on general-purpose graphics hardware, by grouping particle packets into 2D textures and mapping the Boltzmann equations to the rendering operations of the texture units and the frame buffer. They also applied stitching of sub-textures to reduce the overhead of texture switching. They focused on the LBM and its applications to visual simulations of fluids and smoke. Their implementation was extended in 2005 to handle dynamic complex obstacle boundaries [50]. Using this approach, in 2006 Zhu et al. [103] introduced a two-fluid LBM applied to miscible binary mixtures, like pouring honey into water. Due to the simulation of the phase change, the amount of data and floating-point operations were more than doubled and had to start using the newer capabilities of OpenGL, specifically a Frame Buffer Object (FBO) instead of a PBuffer, to achieve real-time simulation. By 2007, Zhao et al. [101] extended previous LBM implementations by including local grid refinement to the LBM. Two grids were used: coarse grid for the global flow behavior and a fine grid for regions of interest. To allow communication between grids, these were rendered as rectangles into textures using FBOs, where the finer grids could access data from the overlapping cells of the coarse grid, and compute their own boundary values using interpolation.

In 2003, Wei et al. [91] presented an approach for modeling the blowing effect of a wind field on light-weight deformable objects immersed within it. The authors accelerated the LBM computation in a similar approach to that of Li et al. [51], extending it with a subgrid model. Their approach was demonstrated using soap bubbles and a feather blown by wind fields; however, it was applicable to other light-weight objects.

In 2004, Fan et al. [24] extended their previous work [71] by simulating the dispersion of contaminants in Times Square using a 30 GPU node cluster. The lattice was decomposed into sub-lattices, each of which was computed on a separate GPU node. The authors proposed extra steps so that those DFs at the borders could be streamed to adjacent nodes.

Qiu et al. [71] described a method for simulating and visualizing the propagation of dispersive contaminants with an application to urban security. They extended their previous work [51] to include the Multiple Relaxation Time Lattice Boltzmann Model (MRTLBM), which has a more complicated collision operation that requires matrix-vector multiplication and can handle complex boundary conditions—such as buildings—efficiently. Zhao et al. [102] incorporated temperature effects directly into this implementation of LBM to simulate the melting and flowing phenomena with different materials in multiple phases. Their modified LBM simulated fluid dynamics and air within a common lattice, to avoid tracking methods for the liquid-air interface.

By 2007, Zhao et al. [100] provided a framework for simulating the natural phenomena related to heat interaction between objects and the surrounding air, by introducing a hybrid thermal Lattice Boltzmann method. The computational approach coupled a MRTLBM with a finite-difference discretization of a standard advection diffusion equation for temperature. Their GPU implementation was based on Li et al. [51], but they saved storage and computation time by defining bounding boxes around the objects and only executed heat transfer computations within these regions.

3.2 Use of GPGPU APIs to perform computations

With the introduction of GPGPU APIs in 2007, considerable improvements to the performance of the LBM were obtainable. By analyzing the parallel portions of the LBM algorithm, and implementing them taking advantage of the APIs capabilities, such as a distinct memory model and specific parallel data access schemes, researchers were able to develop diverse real-time simulations of fluids using the LBM. In this section, works presenting different implementation and optimization strategies for diverse applications are reviewed. Table 2 shows a summary of implementation results of LBM implementations using GPGPU APIs. However, only works that report performance using MLUPS are included in the table, as it is a better indicator of performance for fluid simulations than reporting a speedup or frames per second (FPS) achieved. Additionally, since researchers often tested their approaches on different sized lattices, only the one that offered the best performance is presented.

3.2.1 LBM GPGPU implementations and optimizations

Even though the LBM was successfully implemented using the graphics pipeline, all those applications use a programming style close to the hardware especially developed for graphics applications. With CUDA and similar APIs allowing arbitrary code execution and data access on graphics hardware, it became possible to make full use of their massively parallel architecture. For the case of LBM, in 2008, Tölke [83] presented an efficient implementation of a 2D-Lattice Boltzmann kernel using CUDA. This implementation focused on taking full advantage of coalesced memory accesses, which occur when all threads in a warp access a contiguous chunk of memory [20].

In order to perform coalesced memory accesses, fluid data, in particular DFs, should be aligned such that the N th thread of a block accesses the N th element at address $\text{BaseAddress} + N \cdot \text{Index}$. Index N starts from zero and is local within a block, while BaseAddress is the memory address of the zeroth thread [2]. Additionally, the authors used shared memory to store intermediate results (such as in the streaming step) before writing them to global memory, and thus maintaining alignment for the streaming step, where threads would not store to aligned addresses. However, with the introduction of the Fermi architecture of GPUs and CUDA toolkit version 2.0 in 2008, loading coalesced data manually and storing it temporarily in shared memory for later access did not present an advantage [21, 32, 54].

Tölke extended his prior approach to the D3Q13 model in [84]. Although it could be extended to other discretization stencils such as D3Q15 and D3Q19, memory consumption was so large that it was not supported by the GeForce 8800 Ultra GPU, released in 2006. A specific data layout was also an important consideration for optimization. The data were stored in the structure of arrays (SoA) arrangement, instead of the array of structures (AoS) (first described by Wellein et al. [92] for LBM solvers using CPUs), where values of one distribution for all grid points lie consecutively in memory. Figure 3 illustrates the memory layout for both AoS and SoA approaches. A more detailed description of these data layouts for GPUs can be found in [17].

This approach was later extended to the D3Q19 model by Obrecht et al. [62] in 2011, and minor improvements were made to memory throughput performance

Table 2 Performance obtained for the LBM using GPGPU APIs

References	GPU	Grid size	Stencil	Performance (MLUPS)
Tölke [83]	GeForce 8800 Ultra	3072 ²	D2Q9	481
Tölke and Krafczyk[84]	GeForce 8800 Ultra	128 ² × 512	D3Q13	582
Bailey et al. [4]	NVIDIA 8800 Ultra	160 ³	D3Q19	300
Bernaschi et al. [5]	Tesla C870, 8 GT200 GPUs	1057 × 692 × 1446	D3Q19	955
Kuznik et al. [48]	GTX280	1024 ²	D2Q9	947
Obrecht et al. [61]	GT200	128 ³	D3Q19	516
Feichtinger et al. [25]	22 Tesla S1070	90 × 4500 × 90	D3Q19	17,846
Habich et al. [33]	GTX 280	NA	D3Q19	400
Janßen and Krafczyk [42]	Tesla C1060	64 × 96 × 96	D3Q19	358
Obrecht et al. [62]	GeForce GTX 295	96 ³	D3Q19	512
Schreiber et al. [75]	GeForce GTX 285	64 × 64 × 32	D3Q19	113
Xian and Takayuki [93]	Tesla S1070, 96 × Tesla T10	384 ³	D3Q19	4044
Astorino et al. [2]	GeForce GTX 480	224 ³	D3Q19	370
Rinaldi et al. [72]	GeFoce GTX 260	96 ³	D3Q19	259
Habich et al. [32]	Tesla C2070	200 ³	D3Q19	650
Obrecht et al. [65]	Tesla C1060	192 ³	D3Q19	≈ 2200
Delbosc et al. [21]	Tesla C2070	128 ³	D3Q19	680
Jain et al. [41]	Tesla K20c	256 × 128 × 256	D3Q19	600
Mawson and Revell [54]	Tesla K20c	96 ³	D3Q19	1007
Januszewski and Kostur [43]	Testla K40	254 ³	D3Q19	1400
McIntosh-Smith and Curran [55]	Tesla GTX 780 Ti	128 ³	D3Q19	1734

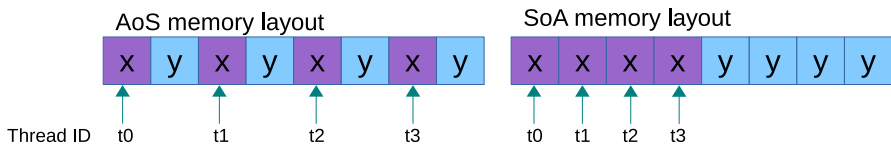


Fig. 3 Storing the data in SoA fashion makes full use of GPU memory bandwidth. Because there is no interleaving of elements of the same field, the SoA layout on the GPU provides coalesced memory accesses and can achieve more efficient global memory utilization

by eliminating misaligned global memory writes through the use of separate arrays for reading and writing distribution functions. However, these improvements came at the cost of doubling device memory usage. Habich et al. [33] presented implementation strategies and optimization approaches for a D3Q19 LBM using CUDA; specifically, they focused on memory alignment and transfers, and register usage. Kuznik et al. [48] proposed a CUDA LBM solver to simulate flow in a lid-driven cavity, showing that single-precision results were nearly indistinguishable from their double-precision counterparts, while performing almost $4 \times$ faster.

In 2009, Bailey et al. [4] used the D3Q19 model to improve upon prior GPU LBM results by increasing GPU multiprocessor occupancy, obtaining an increase of 20% in maximum performance. The authors also tested a two distinct memory access patterns: the A–B pattern, which requires two sets of DFs in memory at all times; and the A–A, which requires only one set of DFs in memory. By using the A–A access pattern, GPU memory requirements were reduced by 50% at a slight detriment in performance, allowing efficient simulation of cubic lattices of up to 160 nodes on the GPU. Another memory usage improvement was introduced by Astorino et al. [2], who presented a modular framework with a high level of generality and reduced the memory requirements of the method. The authors organized the grid as in Tölke's work [84] and proposed an arrangement where the single SoA structure of DFs was split into three independent SoA structures. These structures, together with separate kernels for collision and streaming, allowed them to select the best grid layout dynamically and increase performance. Finally, the swapping technique proposed in [53] resulted in a $0.5 \times$ memory footprint reduction.

Since ordered and fast memory accesses are an important part of CUDA programming, in 2010 Obrecht et al. [61] analyzed the global memory access mechanisms on GPUs, and formulated a model capable of estimating the execution time for a large class of applications. The model was tested on a CUDA-based LBM implementation which led them to multiple optimizations: using the SoA data layout, a two-dimensional grid of one-dimensional blocks, and minimizing the misaligned writes with a pull scheme for the propagation step of the algorithm. The model was able to estimate performance with less than 5% relative error.

Schönherr et al. [74] presented, in 2011, a GPGPU and a multi-core implementation of the LBM for non-uniform grids. Both codes employed second order accurate compact interpolation at the interfaces coupling grids of different resolutions. This implementation allowed a seamless transition from block-structured to almost any unstructured Cartesian grid. The authors also remarked that the limiting factor for the speed of the simulation was the memory bandwidth, rather than the processor performance.

Different optimization approaches were used by Habich et al. [33] to develop a D3Q19 LBM flow solver on NVIDIA GPUs. The correct choice of the data layout, use of shared memory for the propagation, and overall parallelization scheme, as presented by Tölke and Krafczyk [84], were considered. The authors also addressed register usage by introducing a manual indexing scheme: the compiler was forced to reuse the same register again at the small expense of few additional variables. This optimization, however, was due to limitations of the CUDA 1.1 compiler, which lacks

the ability to reuse these registers as they presumably hold information needed in later stages.

By 2013, the authors proposed additional optimization strategies for both GPUs and CPUs in [32]. The stream–collide sequence was chosen in contrast to [33], which only had a minor impact on coalescing and performance. This rearrangement allowed the implementation to avoid the use of shared memory in contrast to [84]. The authors also used the Error Checking and Correction (ECC) feature of Tesla GPUs, which detects and corrects single bit memory errors. Although this feature was necessary to ensure more precise simulations, it led to a performance loss between 30 and 40%. Additionally, the importance of occupancy, as well as optimization strategies to improve overall concurrency, were discussed. Finally, the authors implemented an OpenCL kernel which delivered the same performance on NVIDIA and AMD GPUs and was on par with the CUDA kernels. McIntosh-Smith and Curran [55], as well as Calore et al. [15], obtained a similar result through a LBM code in OpenCL whose performance was highly competitive with the best performance results obtained using the native parallel programming models.

In 2012, Rinaldi et al. [72] presented an optimized LBM implementation focusing primarily on memory access. Instead of using one kernel for each step of the algorithm, they based their approach on a single-step algorithm with a reversed collision–propagation scheme to maximize GPU memory bandwidth, taking advantage of the newer versions of the CUDA programming model and newer NVIDIA GPUs. Their scheme was based on effective access patterns through a single one-dimensional coalesced array, avoiding code branching [17], and by computing the entire LBM algorithm in shared memory, thus limiting the need to access the slower global memory. However, this came at the cost of fewer threads per multiprocessor, as well as the requirement to synchronize threads in a block after each memory load.

Delbosc et al. [21] described a real-time thermal and turbulent flow solver based on the LBM in 2014. The authors took previously mentioned optimizations (such as increasing data coalescence through SoA, and minimizing memory accesses through the use of a single kernel for streaming and collision), and addressed an issue with memory access during the streaming step, previously done through the use of shared memory. It was observed that uncoalesced reads are faster than uncoalesced writes, and thus the pull-in method, which performs uncoalesced reads instead of writes, was used instead of the push-out method, which does the opposite. Figure 4 shows the difference of these methods for a cell. Results with a Tesla C2070 card showed that the pull-in method is about 6% faster than the push-out method. The speedup on this hardware was rather small because the Fermi architecture from NVIDIA uses cached global memory access which tends to hide uncoalesced accesses. Similar optimizations were tested on the Kepler architecture of GPUs by Mawson and Revell [54], who, additionally, showed that the use of shared memory, and an intrinsic memory-less intra-warp shuffle operation, did not improve performance of the streaming step, in spite of the fact that their use increases the number of coalesced accesses to DRAM.

Additional performance optimizations were presented by Tran et al. [85] in 2017. The authors improved cache locality and reduced uncoalesced accesses in the streaming phase of the LBM by tiling the 3D lattice grid into smaller 3D blocks, and by changing the data layout in order to store the data elements in different groups closer

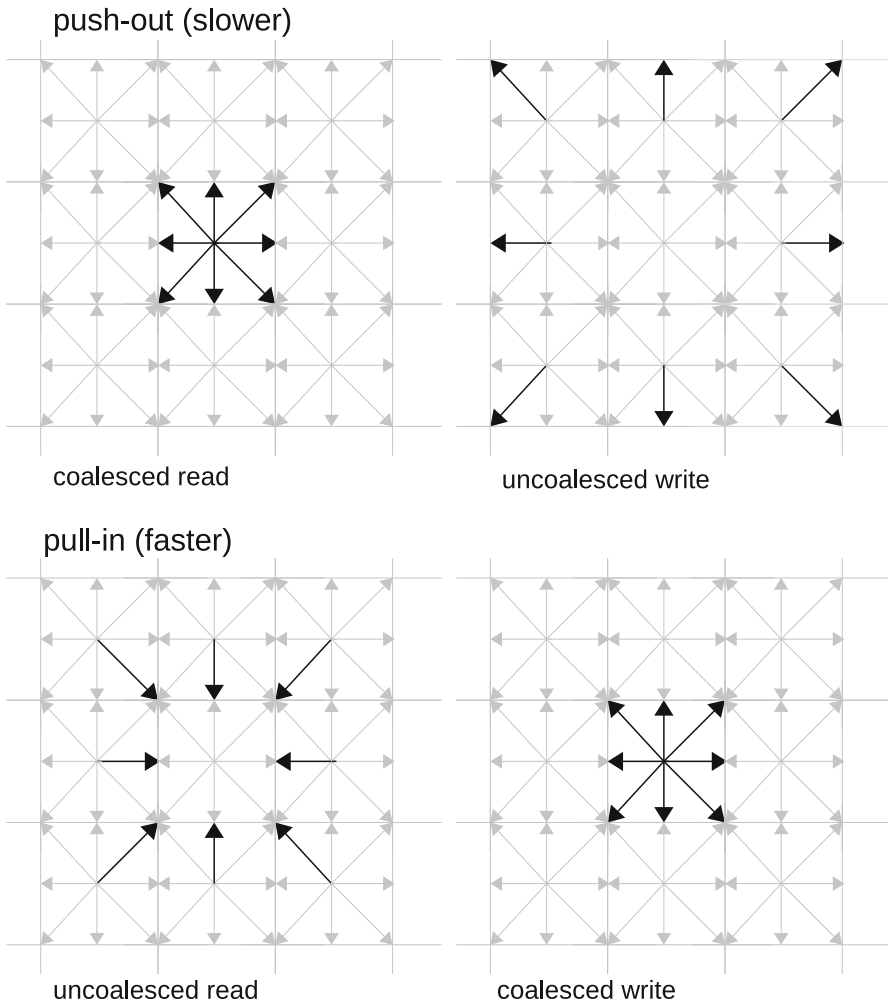


Fig. 4 Pull versus push method for the streaming step of LBM

in the memory. Register usage was also reduced through a series of techniques, such as calculating indexing addresses of distributions manually. Finally, branch divergence at boundary positions was addressed. For boundary positions, a ghost layer was added so that computations would not run out of the index bounds. For the additional actions scenario, if conditions, similar to Listing 1, were combined into other computational statements, as in Listing 2.

Listing 1 if-statement

```
(1) IF(cell type ==
    FLUID)
(2) x = a;
(3) ELSE
(4) x = b;
```

Listing 2 if-removed

```
(1) is fluid = (cell type == FLUID);
(2) x = a * is fluid + b * (!is fluid);
```

More flexible and complete libraries of the LBM have also been introduced. In 2014, Januszewski and Kostur [43] presented *Sailfish*, an open source LBM solver for multiple GPUs using both CUDA and OpenCL. The authors took a novel approach to GPU code implementation by using runtime code generation techniques and a high level, flexible, programming language (Python). Tuning for various types of hardware, and experimentation with different LBM models was also possible. Similarly to Bailey et al. [4], the authors compared the performance for the A–B and A–A memory access patterns. On Tesla-generation devices (GTX 285, Tesla C1060), the A–A memory layout achieved higher performance and was therefore preferred. However, on newer devices the A–B scheme was typically faster, and was ideal when several video memory was available.

Previous works on older GT 200 hardware [61,62] indicated that the cost of unaligned writes was significantly higher than the cost of unaligned reads. However, the authors were unable to replicate this effect on Fermi and Kepler devices (Tesla C2050, K10, K20). This was most likely caused by hardware improvements in the Fermi and later architectures.

All of the previous studies were based either on CUDA or on OpenCL, which may pose severe restriction on the target hardware. Additionally, these two programming standards require substantial changes in the original code, thus threatening the code correctness, portability, and maintainability [14,95]. By using OpenACC's directives on selected code regions, an application can be accelerated with a relative cost of portability versus computing efficiency. To the best of our knowledge, by 2017 there are very few implementations of the LBM using OpenACC.

In 2015, Blair et al. [10] described the porting of a multi-node MPI/OpenMP LBM code to OpenACC. By making minor modifications to their code, the authors were able to increase performance by a factor of 5.5 in a 32 node Cray XC-30 system, each node equipped with a NVIDIA Tesla K40M GPU. Later in 2016, Calore et al. [14] ported a multi-node LBM code using OpenACC, benchmarked it on a variety of processors, and compared the achieved performance with CUDA and OpenCL implementations of the LBM. The authors noted that some major changes in the global structure of code and data organization could not be handled automatically by compilers; specifically, data had to be organized as SoA to allow coalesced performance. Experimental results showed that overall performances were around 50% of what was achievable using CUDA or OpenCL.

Finally, in 2017, Xu et al. [95] studied the performance of fluid flow, heat, and mass transfer simulations based on the LBM. By using the SoA data layout, minimizing memory accesses by combining the stream and collide steps, and by adjusting the level of parallelism through the OpenACC execution model, the authors obtained a speedup between 50 and 60 when compared to the serial implementation of the method.

3.2.2 LBM with free surfaces

Janßen and Krafczyk [42] presented, in 2011, the implementation of a LBM VOF-based algorithm for the simulation of free surface flow problems on GPUs using CUDA. Even though memory improvements [84] were integrated, the free surface part of the algorithm has some performance bottlenecks: it is mainly non-local, the computational domain varies in time and the advection steps apply to the interface nodes only, and non-local write operations have to be performed to ensure a closed interface layer. Nevertheless, the authors still achieved a simulation one order of magnitude faster than comparable CPU implementations.

Schreiber et al. [75] presented an OpenCL implementation of a LBM VOF-based free surface solver capable of running on different architectures. This implementation required special techniques to keep the interface region consistent and avoid race conditions, which was addressed by a novel multipass method instead of using double linked lists to store the interface cells. This method used additional states for each cell (interface to fluid, interface to gas, gas to interface), and extra kernels to address each state correctly. Compared to the basic implementation, only about 20% of its performance was reached due to the required multipass method. Additionally, the authors tested different memory layouts, recommending the A–A pattern for large simulation domains with big memory demand, and the A–B pattern to achieve simulation speed.

An approach to model free surfaces coupling the LBM and the level set method was introduced by Kryza and Dzwinel [47] in 2013. Block configuration was arranged as in [84], with a fixed number of threads. The authors proposed the use of atomic functions to avoid possible race conditions present in the level set method for the correction of the isosurface after the advection step.

In 2014, Jain et al. [41] presented an interactive free surface LBM VOF simulation of generalized Newtonian fluids (GNF) using GPUs. GNFs include regular constant viscosity fluids as well as others, such as blood, which display varying viscosity due to a shifting shear rate. The authors used the SoA data format and coalesced memory accesses to improve performance. Special attention was given to increase kernel occupancy and to reduce thread divergence by sorting cells according to their state.

3.2.3 Usage of GPU clusters for large-scale simulations

Even though the LBM was considerably sped up using GPUs, single-GPU implementations were limited by the available memory of the cards; specifically, limiting the size of the lattices, the stencil used, and the possible applications. Multi-GPU implementations are necessary to run large-scale LBM simulations; however, applications running on multiple GPUs have to face PCI-E bottlenecks, as well as define implementations that minimize inter-GPU communications [66].

In 2010, Bernaschi et al. [5] described the porting of the Lattice Boltzmann component of MUPHY (a software for multi-physics/scale simulations of particles embedded and interacting with fluids) to several GPUs using CUDA. They developed techniques for optimizing the indirect addressing of the LBM lattice nodes for efficient simulations of irregular domains. However, they presented uncoalesced memory accesses,

which reduced their overall performance. Myre et al. [58] created a framework for the integration of LBM GPU code with OpenMP to create applications for multi-GPU clusters. They also examined the performance of the single- and multi-phase LBM. The contributions of various parameters (threads per block, lattice size, memory usage) to the performance were examined and quantified using analysis of variance (ANOVA). They showed that all the LBM simulations primarily depended on simulation geometry and decomposition, and confirmed that the metrics of Efficiency and Utilization were not suitable for memory-bandwidth-dependent codes. These implementations had no issues regarding data communication between CPU and GPU since GPUs were installed in the same node.

By using a multi-node GPU cluster, using 100 out of 680 available GPUs, in 2011 Xian and Takayuki [93] implemented the LBM with CUDA and MPI. Since the data communication must be via CPUs, a data partitioning scheme was used to reduce its size. The main idea was that the size of data for communication decreases with the increase in the number of GPUs used. Another way they used to decrease the communication time was to overlap the computation and the communication by multiple streams.

A fundamental requirement for the efficient use of GPUs in HPC clusters is scalable multi-GPU implementations. Feichtinger et al. [25] presented a multi-GPU LBM solver which uses a block-structured MPI parallelization and was suitable for load balancing and heterogeneous computations on CPUs and GPUs. Although the authors achieved good results, memory access optimizations still had to be included in the fluid solver.

In 2012, Xiong et al. [94] introduced a multi-GPU D2Q9 LBM solver for gas–solid two-phase flow which used up to 576 GPUs on 96 nodes. The authors used the CUDA 3.1 API implementation of streams, which allowed asynchronous execution of kernels. Additionally, the authors achieved the copying of data between a GPU and CPU, and carried out inter-CPU communications, by using OpenMP to launch kernels, and MPI for communication between compute nodes. Their implementation was tested to simulate a gas–solid suspension containing more than one million solid particles and one billion gas lattice cells.

Obrecht et al. [63,66] presented a LBM implementation for GPU clusters which imported and exported data efficiently in each spatial direction, enabling the partition of a 3D domain. Communication between nodes and CUDA kernels was performed with MPI. Obrecht et al. [65] later released a multi-GPU LBM solver based on the well-known D3Q19 MRT model, which was able to perform high-resolution simulations for large Reynolds numbers without facing numerical instabilities.

Potluri et al. [69] addressed the issue of performance bottlenecks that happen when moving data in/out of GPUs. The authors used Inter-Process Communication (IPC), present in CUDA 4.1, to address data movement overheads between processes using different GPUs connected to the same node. Additionally, MPI libraries, like MVA-PICH2, were modified to allow the use of MPI calls directly over GPU device memory. The proposed methods were tested with a CUDA implementation of the LBM for multi-phase flows [73]. After additional CUDA aware MPI implementations became available [45], in 2016 Calore et al. [13] described a massively parallel code for a thermal LBM on later generations of GPUs.

In 2015, Feichtinger et al. [26] extended the waLBerla software framework, specialized in multi-physics simulations centered around the LBM, to consider simulations on CPU–GPU clusters, and focused on maintainability and performance. The software supports a pure-MPI and a hybrid parallelization approach capable of heterogeneous simulations using CPUs and GPUs in parallel. Weak and strong scaling performance results were obtained on the Tsubame 2.0 cluster [40] for more than 1000 GPUs.

4 Applications of GPU-based LBM

Since CFD simulations are computationally intensive, in prior hardware generations they had to be precomputed to present an animation later. Certain effects such as water, smoke, and fire were not practical to include in applications. Additionally, simulations that tried to compute large domains were computationally expensive and time consuming, and changes to the simulation's parameters usually required to rerun the simulation entirely. Interactive applications would not have been able to achieve real-time processing without the introduction of GPUs. Here, we present several applications that use the LBM in different fields, and that benefit from using GPUs for processing.

4.1 Gas simulations

Some of the first applications that used GPUs were intended for gas, steam, and smoke. Li et al. [51] presented, in 2003, applications such as smoke emanating from a chimney, or steam rising from a teapot. Simulation results were visualized by either directly showing the color-encoded velocity field or by injecting mass-less particles into the velocity field from an inlet. The gas was then rendered with texture splats. Qiu et al. [71] simulated the propagation of contaminants within an urban scene; in particular, scenes characterized by skyscrapers and deep urban canyons. This application could help understand meteorological, and fluid dynamic processes governing dispersion in urban areas and also allow emergency management personnel to adequately plan and respond to potential accidents or attacks involving toxic airborne contaminants. This work was later expanded to consider a large urban scene by Fan et al. [24].

4.2 Object deformation

An application where a flow field interacted with deformable objects immersed within it was presented by Wei et al. [91] in 2003. Their approach simulated soap bubbles and feathers blown by wind fields. The feather floated and fluttered in response to lift and drag forces of the wind, and the single bubble simulation allowed the user to directly interact with the wind field, and thereby influence the dynamics in real time.

Another example of object deformation was presented by Zhao et al. [102], who simulated melting, and flowing phenomena with different materials in multiple phases. Solid objects were melted because of heating and the melted liquid flowed while interacting with the ambient air flow. The modified LBM modeled the fluid dynamics of the air flow and the melted liquid within a common lattice framework.

4.3 Thermal models

In 2007, Zhao et al. [100] provided an interactive physically based approach for simulating the natural phenomena related to heat interaction between objects and the surrounding air. The heat distribution of the objects was represented by a novel temperature texture, and the thermal flow dynamics that models the air flow interacting with the heat were modeled by a hybrid thermal Lattice Boltzmann model (HTLBM). Additionally, a nonlinear ray tracing method to render the visual results of these phenomena was presented.

Bertazzo et al. [7] introduced, in 2012, a new thermal LB model using a D2Q37 stencil, which was able to correctly describe compressible thermal fluids, including combustion effects. The authors tested the model on a multi-GPU cluster with Tesla GPUs. Biferale et al. [8] described a thermal compressible LBM, while also focusing on the more complex D2Q37 stencil, to correctly describe compressible thermal fluids that obeyed the equation-of-state of a perfect gas. Kraus et al. [46] used the D2Q37 model as a significant benchmark to analyze the performance of newer GPU boards based on the Fermi and Kepler architectures, since critical kernels in this code require both high memory bandwidth on sparse memory addressing patterns and floating-point throughput. A thermal LBM code for GPU clusters was presented by Calore et al. [13].

Mynam et al. [57] proposed a new method to simulate non-isothermal flows, by coupling the LBM with the finite-difference scheme for the temperature field. The algorithm was used to solve the flow in buoyancy-driven cavity problems and verify the validity of the algorithm by benchmarking the thermal flow patterns with known results.

In 2013, Obrecht et al. [64] presented a single-node multi-GPU thermal Lattice Boltzmann solver. The authors implemented a hybrid model which combined MRTLBM with a finite-difference method for temperature. Using appropriate hardware, the proposed program was able to run with up to eight GPUs in parallel. With the then available GPUs, it was possible to perform simulations on lattices containing as much as 3.2×10^8 nodes.

4.4 Handling turbulence

Li et al. [49] proposed, in 2012, a multi-GPU simulation of Large Eddy Simulation (LES), a mathematical model for turbulence used in CFD. The authors tested their approach with an implementation of two-dimensional lid-driven cavity flow at high Reynolds number, which was capable of running on four Fermi GPUs in parallel. In 2013, Tanno et al. [78] implemented three methods: the Lattice Boltzmann method, pseudospectral method, and artificial compressibility method; and calculated homogeneous isotropic turbulent flows. Their results proved that the Lattice Boltzmann method on a GPU has advantages of accuracy and computational speed, while yielding similar flow fields to other methods.

Ye and Li [96] presented an efficient implementation strategy for entropic Lattice Boltzmann method (ELBM), a modified LBM used for the stable computational sim-

ulation of high Reynolds number fluid flows. In general, this stability was gained at the price of some computational overhead associated with the requirement of adjusting the local relaxation parameter of the standard LBM. The authors proposed GPU algorithms which address this concern.

4.5 Modifying physical properties

Miscible fluid mixtures, like pouring honey into water, are common phenomena in daily life. While two miscible fluids are mixed together, their appearances in terms of colors and shapes change due to their mixing interaction. In 2006, Zhu et al. [103] introduced a real-time two-fluid LBM, called TFLBM, applied to miscible binary mixtures. The viscous and diffusing properties of the fluid were considered separately, so that the physical insight was exposed more clearly. Bernaschi et al. [6] proposed, in 2009, an implementation on a GPU architecture of LBM soft-glass flowing systems. Their GPU version is shown to provide more than an order of magnitude in elapsed time over the corresponding CPU version.

The LBM is often advocated as an effective simulation tool for multi-phase flow in porous media. In 2013, Li et al. [52] extended the mainstream Rothman–Keller (R–K) multi-phase model with a multi-relaxation-time scheme by adding perturbation in the moment space and a diffusion process. The method was validated by numerical experiments on the Laplace’s law and dynamics of spreading, and the drainage and imbibition processes of real cores were then simulated.

Tripathi and Narayanan [86] presented an approach to simulate both Newtonian and GNF the LBM. The method can model the macroscopic behavior of such fluids by simulating the variation of properties such as viscosity through the bulk of the fluid. Additionally, the change in viscosity of a GNF and its free surface interactions with obstacles and boundaries were simulated.

4.6 Medical simulations

Melchionna et al. [56] introduced in 2010 a method for clinical cardiovascular diagnosis based on accurate simulation of cardiovascular blood flow. Additionally, a procedure for the analysis of real-life cardiovascular blood flow case studies, namely, anatomic data acquisition, geometry and mesh generation, flow simulation and data analysis and visualization was presented.

In 2012, Bisson et al. [9] presented a multi-GPU implementation of MUPHY for hemodynamic simulations in anatomically realistic geometries. The solution couples a LBM representation of the blood plasma, with suspended biological bodies as particles, such as red blood cells. This work was also one of the first examples of high-performance solutions of multi-scale physics and bio-fluid applications in realistic geometries.

In 2013, Nita et al. [60] focused on the application of a double-precision implementation of the LBM for patient-specific blood flow computations. An indirect addressing scheme was used to reduce the memory requirements. Three GPU cards were evaluated with different 3D benchmark applications: Poiseuille flow, lid-driven cavity

flow, and flow in an elbow-shaped domain. The GTX680 card was determined as best performing GPU and was subsequently used to compute blood flow in an aorta geometry with coarctation.

Radio-frequency ablation (RFA), one of the most widely used minimally invasive ablative therapy of liver cancer, is challenged by, among others, the presence of blood vessels and time-varying thermal diffusivity that make the prediction of the extent of ablated tissue difficult. Audigier et al. [3] proposed a new model of the biological mechanisms involved in RFA of abdominal tumors based on the LBM to predict the extent of ablation. The GPU implementation showed a speedup of 45 with respect to single-core implementation of LBM.

By 2015, Wang et al. [90] introduced a GPU version of a variation of the LBM: volumetric Lattice Boltzmann method (VLBM). The VLBM does not need to differentiate fluid and boundary cells, so branching in GPU kernels was minimized and execution was accelerated. The authors tested their method with a pulsatile blood flow in a patient-specific carotid artery segmented from an anonymous clinical CT image and achieved more than 30 times speedup over the CPU version.

In 2015, Campos et al. [16] described an optimized implementation of the LBM for computational simulations of the cardiac electrical activity in 3D domains, with regular and complex geometries, using the mono-domain model. The method uses a collision model with multiple relaxation parameters in order to consider the anisotropy of the cardiac tissue. With near-real-time simulations in a single computer equipped with a modern GPU, these results showed that the proposed framework is a promising approach for application in a clinical workflow.

4.7 General use

The LBM has proven to be versatile enough to be used in fields not necessarily tied to CFD. One of such examples is the one introduced by Zhao [99] in 2008, who proposed a GPU-accelerated PDE solver based on the LBM. The solver was used to solve the elliptic Laplace and Poisson equations with a diffusion process. These PDEs are widely used in modeling and manipulating images, surfaces, and volumetric data sets. This method was applied to several examples in image processing, computer graphics, and visualization, such as volume smoothing, surface fairing, and image editing, achieving outstanding performance on contemporary graphics hardware.

LBM simulations were also used for applications that could be related to art creation, such as the work by Chu and Tai [18] in 2005, who presented a physically based method for simulating ink dispersion in absorbent paper. The authors devised a novel fluid flow model suitable for simulating percolation in disordered media, like paper, in real time.

In 2010, Geveler et al. [29] presented a combination of the shallow water equations (SWE) and the LBM, with methods to stabilize dry-states as well as for fluid-structure interaction, to simulate ‘real-world’ free surface including their interaction with moving rigid bodies. This approach was implemented on a GPU MPI-based cluster. Tubbs and Tsai [87] described a generalized Lattice Boltzmann equation (GLBE) with a MRT collision method to simulate shallow water flow. A two-relaxation-time (TRT) method with two speed-of-sound techniques is used to solve the advection–dispersion

equation. The dam-break problem demonstrated that the MRTLBM was able to handle complex flow structures, such as shocks, rarefaction waves, and contact discontinuities.

Amplified perception of liquids through the use of haptic technology was presented by Zhang et al. [98] in 2016, who introduced a multimodal visualization and steering system with visual and haptic interfaces for a real-time GPU-based coupled fluid-structure simulation. Both the LBM and the haptic updating were accelerated with GPU. Furthermore, a model of palpation was proposed to allow users to touch, push and sense the dynamic fluid motion inside a deformable tube. These solutions could be integrated into virtual reality simulations to increase immersion and improve the user's experience.

5 Challenges and research directions

There are still several challenges regarding the implementation of GPU-based LBM solutions. The following are some of the most prevalent:

- *Usage of consumer grade GPUs* One of the main issues when developing GPU applications is the selection of the graphics card. Most of the presented works used the Geforce line of GPUs by Nvidia, most likely because of pricing and availability. Even though the Tesla line of GPUs from NVIDIA, or the Radeon Vega from AMD, are the ones designed for computing intensive tasks, and usually yielding more accurate results, their higher price limits their widespread use.
- *Constant hardware and API releases* Another challenge to overcome is the constant release of newer hardware and APIs. Each new generation of GPUs and APIs usually comes with newer capabilities and features, sometimes rendering previous results invalid. For example, with the release of CUDA 6 came the Unified Memory model, which makes it easy to allocate and access data that can be used by code running on any processor in the system, CPU or GPU. This could potentially help streamline GPU-based CFD simulation development, and to the best of our knowledge, has yet to be used. Nvidia just released version 9 of its API, with features such as thread management with groups, and thread reutilization. Both features could be explored to improve GPU-based LBM simulations.
- *Data layout* One common issue with GPU simulations is that data has to be laid out in specific ways to improve performance. Specifically, the SoA data layout has proven to yield the best results for LBM simulations. This usually means that fluid simulation data have to be prepared before it can be used, and the problem has to be rethought to include the new layout. This is somewhat due to how GPUs process data. With newer hardware advances and API being released, the way in which data are laid out could be reworked. Usage of newer GPU features, such as Unified Memory, could help ease this requirement.
- *Memory consumption* Even though GPUs provide a platform for more efficient computation of complex problems, their memory capacity is reduced when compared to their CPU counterparts, and expensive memory transfers are necessary to deal with larger problems. One of the main challenges for GPU solutions of the LBM is its large memory requirements, which in turn limits the simulation

of larger domains. To counter this, authors have used GPU clusters, and have taken advantage of the different memory schemes present in GPUs. More recently, Valero-Lara [88] presented two new techniques to help reduce memory requirements while keeping high performance in large simulations.

- *Cloud solutions* Even though GPU clusters have been used to solve larger LBM simulations, their scalability is limited by the amount available nodes. One possible direction is the use of services such as Google Cloud Platform or Amazon Web Services, both of which provide GPU instances, to compute large-scale fluid simulations. This would allow researchers to obtain the right amount of computing power for their specific problems, without the need to acquire and maintain expensive GPU data centers.

Even though there are still challenges to overcome, the use of GPU-based LBM solutions is still being researched in different areas. Instead of focusing on specific domain solutions using the LBM, the following areas have promising results for both performance and applications:

- *Boundless and Meshfree LBM* Wang et al. [89] presented a free flow simulation of the LBM in irregular areas using a boundless grid. The authors used a method called Hierarchical runlength encoding to simulate a large-scale flow. This allowed them to expand the grid dynamically according to the fluid flow, and only use computational resources proportional to the volume of the fluid. However, their solution only used GPUs for a realistic visualization of the fluid flow. Another potential area of research is the use of GPUs for meshfree simulations of the LBM [79].
- *Use of artificial intelligence* As was previously discussed, CFD simulations are computationally expensive, consume large amounts of memory, and their solution is time consuming. Even though GPUs have been used to achieve faster simulations, said problems are still present and have been tackled through different means. A relatively unused technique to solve these drawbacks is the use of Artificial Intelligence, specifically through deep learning. Guo et al. [31] in 2016, used Convolutional Neural Networks (CNN) to approximate a general and flexible model for real-time prediction of non-uniform steady flow in a 2D or 3D domain. The authors showed that CNN estimated the velocity field of a fluid two orders of magnitude faster than a GPU-accelerated CFD solver at a cost of a low error rate. Another approach to reduce the memory usage of CFD simulations through convolutional autoencoders was presented by Hennigh [37]. The result was a memory efficient neural network that could be used to reproduce fluids with large grid sizes and complex geometries while maintaining accuracy. Additionally, Nvidia has recently focused on improving the performance of their GPUs for Artificial Intelligence, specially for deep learning.
- *Mobile solutions* Most of the work that has been introduced has focused on desktop GPUs. However, the release of the Tegra GPUs by Nvidia opens the possibility of developing CFD simulations for mobile devices. This would allow the use of fluid simulations in more diverse fields, such as in teaching, or for Augmented Reality applications. Hardwood and Revell [35] presented CFD simulations on mobile devices, specifically for devices with a Tegra K1 GPU. The authors integrated

CUDA, C++, JNI and Java into a typical Android app, while reporting better throughput and power consumption when compared to a CPU implementation.

6 Discussion and conclusions

We have reviewed the progress made in developing GPU-based LBM fluid solvers, focusing on optimization and performance enhancement approaches taken. Additionally, we have presented several applications which showcase the versatility of the method.

The main drawback that rapidly appears when implementing the LBM on GPU is the memory bandwidth, i.e., accessing data in the GPU memory takes longer than the actual computations on these data. That is why authors have focused on increasing the memory bandwidth in order to implement an optimized LBM program. An easy and efficient way to increase the memory throughput of a program is to reduce the number of accesses to global memory by avoiding redundant accesses to data. It was proved that using a single kernel for streaming and colliding yielded better results than using two kernels, as only one read and one write to the distribution functions is required [21, 32, 72, 95].

Another important optimization strategy for memory access, if not the most important, is to arrange data such that memory accesses are coalesced. This is mainly achieved by using the SoA data layout. Additionally, it is recommended to avoid uncoalesced writes, which are present during the streaming step of the algorithm, by using a pull-in method instead of a push-out method. Another important consideration is that the use of shared memory for the streaming step may have been useful for older GPUs and APIs, but several studies [21, 32, 54, 72] performed on newer frameworks found that shared memory no longer provided an improved performance of the streaming step, in spite of the fact that their use increases the number of coalesced accesses to VRAM.

Although free surface flows also benefit from these optimizations, they present other challenges that still limit their performance. The main issue is around interface cells, as mentioned in [41, 42, 75]. Since some steps for free surface LBM are performed only for those cells, and the kernels are called for all cells, thread divergence in the kernels is introduced. Additionally, kernels have been proved to achieve only 75% occupancy [41], which in turn limits the maximum performance. Some optimizations, such as the A–A and the A–B memory patterns, the use of atomic operations, and minimizing thread divergence by sorting cells according to their state have been proposed. However, these are still not optimal, and there is still room for improvement.

Large-scale multi-GPU implementations are usually based on using MPI for communication between compute nodes. However, the main issues are performance bottlenecks that happen when moving data in/out of GPUs, since the data communication was made via CPUs. The authors found ways to minimize CPU–GPU communication through different means, such as asynchronous execution of kernels through streams [94], overlapping the computation and the communication by multiple streams [93], using IPC to address data movement overheads between processes using different GPUs [69], and using CUDA aware MPI implementations [13]. It is

important to note that some of these solutions were not available until later API versions were released.

GPU architecture and API version play an important role in LBM implementations, optimization, and performance. The first attempts at implementing LBM GPU-based solvers differ from the more recent presented works; even approaches introduced more closely to each other could also differ considerably. This could be explained by considering that GPU usage and development has had a consistent growth over the years, and newer cards and software frameworks provide different features which help address previously found issues; such as Fermi cards from NVIDIA hiding uncoalesced memory accesses [21]. As could be seen in several solutions, new features may drive different methods and breakthroughs, and previous results may improve depending on the generation of the GPU and API version used.

Although it was proven that OpenCL kernels delivered the same performance on NVIDIA and AMD GPUs and were on par with the CUDA kernels [15, 32, 55], accelerating the LBM with alternative programming models, such as OpenACC, still does not show the same performance as with native parallel programming models. Still, achieving lower performance by trading better programmability with computing efficiency, may be considered a satisfactory result [14].

Finally, several challenges to overcome were discussed, mainly regarding the use of specific hardware and APIs, and memory consumption of the simulations. Additionally, some new areas that are further improving the LBM, such as with the use of Artificial Intelligence, or that are using the method for diverse applications, were also mentioned.

Acknowledgements The authors would like to thank the Tecnológico de Monterrey IT and Computer Department for its support. This work was supported, in part, by the 2015 Google Faculty Research Awards and Tides Foundation under Grant No. TFR15-00145, and by the Consejo Nacional de Ciencia y Tecnología (CONACYT) under Grant No. 342814.

References

1. Aristov VV, Frolova AA, Zabelok SA (2004) Parallel algorithms of direct solving the Boltzmann equation in aerodynamics problems. Elsevier, Amsterdam
2. Astorino M, Becerra Sagredo J, Quarteroni A (2012) A modular lattice Boltzmann solver for GPU computing processors. *SeMA J* 56(59):53–78
3. Audigier C, Mansi T, Delingette H, Rapaka S, Mihalef V, Sharma P, Carnegie D, Boctor E, Choti M, Kamen A et al (2013) Lattice Boltzmann method for fast patient-specific simulation of liver tumor ablation from CT images. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Berlin, pp 323–330
4. Bailey P, Myre J, Walsh SD, Lilja DJ, Saar MO (2009) Accelerating lattice Boltzmann fluid flow simulations using graphics processors. In: Parallel Processing, 2009. ICPP'09. International Conference on. IEEE, pp 550–557
5. Bernaschi M, Fatica M, Melchionna S, Succi S, Kaxiras E (2010) A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. *Concurr Comput Pract Exp* 22(1):1–14
6. Bernaschi M, Rossi L, Benzi R, Sbragaglia M, Succi S (2009) Graphics processing unit implementation of lattice Boltzmann models for flowing soft systems. *Phys Rev E* 80(6):066,707
7. Bertazzo A, Mantovani F, Pivanti M, Pozzati F, Schifano SF, Tripiccone R (2012) Implementation and optimization of a thermal lattice Boltzmann algorithm on a multi-GPU cluster. In: 2012 innovative parallel computing, InPar 2012

8. Biferale L, Mantovani F, Pivanti M, Pozzati F, Sbragaglia M, Scagliarini A, Schifano SF, Toschi F, Tripiccione R (2013) An optimized D2Q37 lattice Boltzmann code on GP-GPUs. *Comput Fluids* 80(1):55–62. <https://doi.org/10.1016/j.compfluid.2012.06.003>
9. Bisson M, Bernaschi M, Melchionna S, Succi S, Kaxiras E (2012) Multiscale hemodynamics using GPU clusters. *Commun Comput Phys* 11(01):48–64
10. Blair S, Albing C, Grund A, Jocksch A (2015) Accelerating an MPI lattice Boltzmann code using OpenACC. In: *Proceedings of the second workshop on accelerator programming using directives*. ACM, p 3
11. Brandvik T, Pullan G (2007) Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware. *Proc Inst Mech Eng Part C J Mech Eng Sci* 221(12):1745–1748
12. Brandvik T, Pullan G (2008) Acceleration of a 3D Euler solver using commodity graphics hardware. In: *46th AIAA aerospace sciences meeting and exhibit*, pp 1–15
13. Calore E, Gabbana A, Kraus J, Pellegrini E, Schifano S, Tripiccione R (2016) Massively parallel lattice-Boltzmann codes on large GPU clusters. *Parallel Comput* 58:1–24
14. Calore E, Gabbana A, Kraus J, Schifano SF, Tripiccione R (2016) Performance and portability of accelerated lattice Boltzmann applications with OpenACC. *Concurr Comput Pract Exp* 28(12):3485–3502
15. Calore E, Schifano SF, Tripiccione R (2014) A portable OpenCL lattice Boltzmann code for multi-and many-core processor architectures. *Proc Comput Sci* 29:40–49
16. Campos J, Oliveira RS, dos Santos RW, Rocha BM (2016) Lattice Boltzmann method for parallel simulations of cardiac electrophysiology using GPUs. *J Comput Appl Math* 295:70–82
17. Cheng J, Grossman M, McKercher T (2014) *Professional CUDA C programming*. Wiley, New York
18. Chu NSH, Tai CL (2005) MoXi: real-time ink dispersion in absorbent paper. *ACM Trans Graph*. 24(3):504–511. <http://visgraph.cs.ust.hk/MoXi/>
19. Clough D (2014) Lattice Boltzmann liquid simulations on graphics hardware, Ph.D. thesis. University of Cape Town
20. Corporation N (2016) Parallel programming and computing platform. http://www.nvidia.com/object/cuda_home_new.html. Accessed 11 May 2016
21. Delbosco N, Summers JL, Khan A, Kapur N, Noakes CJ (2014) Optimized implementation of the lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation. *Comput Math Appl* 67(2):462–475
22. Elizharova T, Milyukova OY (2004) Parallel algorithm for numerical simulation of 3D incompressible flows. Elsevier, Amsterdam
23. Evans MW, Harlow FH, Bromberg E (1957) The particle-in-cell-method for hydrodynamic calculations, Technical report. DTIC Document
24. Fan Z, Qiu F, Kaufman A, Yoakum-stover S (2004) GPU cluster for high performance computing. *IEEE Supercomput* 00(1):47
25. Feichtinger C, Habich J, Köstler H, Hager G, Rüde U, Wellein G (2011) A flexible patch-based lattice Boltzmann parallelization approach for heterogeneous GPU–CPU clusters. *Parallel Comput* 37(9):536–549
26. Feichtinger C, Habich J, Köstler H, Rüde U, Aoki T (2015) Performance modeling and analysis of heterogeneous lattice Boltzmann simulations on CPU–GPU clusters. *Parallel Comput* 46:1–13
27. Ferziger JH, Peric M (2012) *Computational methods for fluid dynamics*. Springer, Berlin
28. Gaudlitz D, Landmann B, Indinger T (2013) Accelerated CFD simulations using Eulerian and Lagrangian methods on GPUs. *Proc Eng* 61:392–397
29. Geveler M, Ribbrock D, Göddeke D, Turek S (2010) Lattice-Boltzmann simulation of the shallow-water equations with fluid–structure interaction on multi-and manycore processors. In: *Facing the multicore-challenge*. Springer, Berlin, pp 92–104
30. Goodnight N, Lewin G, Luebke D, Skadron K (2003) A multigrid solver for boundary value problems using programmable graphics hardware. In: *ACM SIGGRAPH 2005 Courses*, p 193
31. Guo X, Li W, Iorio F (2016) Convolutional neural networks for steady flow approximation. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp 481–490
32. Habich J, Feichtinger C, Köstler H, Hager G, Wellein G (2013) Performance engineering for the lattice Boltzmann method on GPGPUs: architectural requirements and performance results. *Comput Fluids* 80(1):276–282. <https://doi.org/10.1016/j.compfluid.2012.02.013>

33. Habich J, Zeiser T, Hager G, Wellein G (2011) Performance analysis and optimization strategies for a D3Q19 lattice Boltzmann kernel on nVIDIA GPUs using CUDA. *Adv Eng Softw* 42(5):266–272
34. Harris MJ (2004) GPU gems—chapter 38. Fast fluid dynamics simulation on the GPU. *GPU Gems 3*. https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch38.html. Accessed 04 May 2017
35. Harwood AR, Revell AJ (2018) Interactive flow simulation using Tegra-powered mobile devices. *Adv Eng Softw* 115:363–373
36. He X, Luo LS (1997) Lattice Boltzmann model for the incompressible Navier–Stokes equation. *J Stat Phys* 88(3–4):927–944
37. Hennigh O (2017) Lat-net: compressing lattice Boltzmann flow simulations using deep neural networks. arXiv preprint [arXiv:1705.09036](https://arxiv.org/abs/1705.09036)
38. Hirt CW, Nichols BD (1981) Volume of fluid (VOF) method for the dynamics of free boundaries. *J Comput Phys* 39(1):201–225
39. Holt M (2012) Numerical methods in fluid dynamics. Springer, Berlin
40. Information GS, Center C (2017) Tsubame2. <http://www.gsic.titech.ac.jp/en/tsubame2>. Accessed 04 May 2017
41. Jain S, Tripathi N, Narayanan PJ (2014) Interactive simulation of generalised Newtonian fluids using GPUs. In: Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing. ACM, p 79
42. Janßen C, Krafczyk M (2011) Free surface flow simulations on GPGPUs using the LBM. *Comput Math Appl* 61(12):3549–3563. <https://doi.org/10.1016/j.camwa.2011.03.016>
43. Januszewsky M, Kostur M (2014) Sailfish, a flexible multi-GPU implementation of the lattice Boltzmann method. *Comput Phys Commun* 185(9):2350–2368
44. Jie T, XuBo Y (2009) Physically-based fluid animation: a survey. *Sci China Ser F Inf Sci* 52(5):723–740
45. Kraus J (2013) An introduction to CUDA-aware MPI. <http://developer.nvidia.com/content/introduction-cuda-aware-mpi>. Accessed 10 March 2017
46. Kraus J, Pivanti M, Schifano SF, Tripiccione R, Zanella M (2013) Benchmarking GPUs with a parallel lattice-Boltzmann code. In: Proceedings of the symposium on computer architecture and high performance computing, pp 160–167
47. Kryza T, Dzwiniel W (2013) Coupling lattice Boltzmann gas and level set method for simulating free surface flow in GPU/CUDA environment. In: International Conference on Parallel Processing and Applied Mathematics. Springer, Berlin, pp 731–740
48. Kuznik F, Obrecht C, Rusaouen G, Roux JJ (2010) LBM based flow simulation using GPU computing processor. *Comput Math Appl* 59(7):2380–2392
49. Li Q, Zhong C, Li K, Zhang G, Lu X, Zhang Q, Zhao K, Chu X (2012) Implementation of a lattice Boltzmann method for large eddy simulation on multiple GPUs. In: 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, pp 818–823. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6332253>
50. Li W, Fan Z, Wei X, Kaufman A (2005) GPU-based flow simulation with complex boundaries. In: *GPU Gems 2*. https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter47.html. Accessed 25 Apr 2018
51. Li W, Wei X, Kaufman A (2003) Implementing lattice Boltzmann computation on graphics hardware. *Vis Comput* 19(7–8):444–456. <https://doi.org/10.1007/s00371-003-0210-6>
52. Li X, Zhang Y, Wang X, Ge W (2013) GPU-based numerical simulation of multi-phase flow in porous media using multiple-relaxation-time lattice Boltzmann method. *Chem Eng Sci* 102:209–219
53. Mattila K, Hyväluoma J, Rossi T, Aspñäs M, Westerholm J (2007) An efficient swap algorithm for the lattice Boltzmann method. *Comput Phys Commun* 176(3):200–210
54. Mawson MJ, Revell AJ (2014) Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture nVidia GPUs. *Comput Phys Commun* 185(10):2566–2574
55. McIntosh-Smith S, Curran D (2014) Evaluation of a performance portable lattice Boltzmann code using OpenCL. In: Proceedings of the international workshop on OpenCL 2013 & 2014. ACM
56. Melchionna S, Bernaschi M, Succi S, Kaxiras E, Rybicki FJ, Mitsouras D, Coskun AU, Feldman CL (2010) Hydrokinetic approach to large-scale cardiovascular blood flow. *Comput Phys Commun* 181(3):462–472

57. Mynam M, Sahasrabudhe N, Nandgaonkar A (2012) GPU implementation of a novel hybrid lattice Boltzmann method for non-isothermal flows. In: Proceedings of the 5th ACM Compute Conference: Intelligent & Scalable System Technologies. ACM, p 7
58. Myre J, Walsh SD, Lilja D, Saar MO (2011) Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters. *Concurr Comput Pract Exp* 23(4):332–350
59. Niemeyer KE, Sung CJ (2014) Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. *J Supercomput* 67(2):528–564
60. Nita C, Itu LM, Suciuc C (2013) GPU accelerated blood flow computation using the lattice Boltzmann method. In: High Performance Extreme Computing Conference (HPEC), 2013 IEEE, pp 1–6
61. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2010) Global memory access modelling for efficient implementation of the lattice Boltzmann method on graphics processing units. In: International Conference on High Performance Computing for Computational Science. Springer, Berlin, pp 151–161
62. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2011) A new approach to the lattice Boltzmann method for graphics processing units. *Comput Math Appl* 61(12):3628–3638
63. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2011) The helma project: multi-GPU implementation of the lattice Boltzmann method. *Int J High Perform Comput Appl* 23(3):295–303
64. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2013) Multi-GPU implementation of a hybrid thermal lattice Boltzmann solver using the TheLMA framework. *Comput Fluids* 80:269–275
65. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2013) Multi-GPU implementation of the lattice Boltzmann method. *Comput Math Appl* 65(2):252–261
66. Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2013) Scalable lattice Boltzmann solvers for CUDA GPU clusters. *Parallel Comput* 39(6–7):259–270
67. Osher S, Fedkiw RP (2001) Level set methods: an overview and some recent results. *J Comput Phys* 169(2):463–502
68. Posey S (2013) Considerations for GPU acceleration of parallel CFD. *Proc Eng* 61:388–391
69. Potluri S, Wang H, Bureddy D, Singh AK, Rosales C, Panda DK (2012) Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication. In: Parallel and distributed processing symposium workshops & PhD forum (IPDPSW), 2012 IEEE 26th international. IEEE, pp 1848–1857
70. Qian Y, d’Humières D, Lallemand P (1992) Lattice BGK models for Navier–Stokes equation. *EPL (Europhys Lett)* 17(6):479
71. Qiu FQF, Zhao YZY, Fan ZFZ, Wei XWX, Lorenz H, Wang JWJ, Yoakum-Stover S, Kaufman A, Mueller K (2004) Dispersion simulation and visualization for urban security. In: IEEE visualization 2004, pp 553–560. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1372242
72. Rinaldi PR, Dari EA, Vénere MJ, Clause A (2012) A lattice-Boltzmann solver for 3D fluid simulation on GPU. *Simul Modell Pract Theory* 25:163–171. <https://doi.org/10.1016/j.simpat.2012.03.004>
73. Rosales C (2011) Multiphase LBM distributed over multiple GPUs. In: Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, pp 1–7
74. Schönherr M, Kucher K, Geier M, Stiebler M, Freudiger S, Krafczyk M (2011) Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs. *Comput Math Appl* 61(12):3730–3743
75. Schreiber M, Neumann P, Zimmer S, Bungartz HJ (2011) Free-surface lattice-Boltzmann simulation on many-core architectures. *Proc Comput Sci* 4:984–993. <https://doi.org/10.1016/j.procs.2011.04.104>
76. Stam J (1999) Stable fluids. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co, pp 121–128
77. Tan J, Yang X (2009) Physically-based fluid animation: a survey. *Sci China Ser F Inf Sci* 52(5):723–740
78. Tanno I, Hashimoto T, Yasuda T, Tanaka Y, Morinishi K, Satofuka N (2013) Simulation of turbulent flow by lattice Boltzmann method and conventional method on a GPU. *Comput Fluids* 80(1):453–458. <https://doi.org/10.1016/j.compfluid.2012.01.011>
79. Tanwar S (2018) A meshfree-based lattice Boltzmann approach for simulation of fluid flows within complex geometries: application of meshfree methods for LBM simulations. In: Analysis and applications of lattice Boltzmann simulations. IGI Global, pp 188–222
80. Thürey N (2007) Physically based animation of free surface flows with the lattice-Boltzmann method, Ph.D. thesis. University of Erlangen-Nuremberg

81. Thürey N, Rüde U (2004) Free surface lattice-Boltzmann fluid simulations with and without level sets. In: *Vision, modeling, and visualization 2004: proceedings*, Stanford, p 199
82. Thürey N, Rüde U (2009) Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids. *Comput Vis Sci* 12(5):247–263
83. Tölke J (2008) Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by nVIDIA. *Comput Vis Sci* 13(1):29–39
84. Tolke J, Krafczyk M (2008) Teraflop computing on a desktop PC with GPUs for 3D CFD. *Int J Comput Fluid Dyn* 22(7):443–456
85. Tran NP, Lee M, Hong S (2017) Performance optimization of 3D lattice Boltzmann flow solver on a GPU. *Sci Program* 2017:1205892. <https://doi.org/10.1155/2017/1205892>
86. Tripathi N, Narayanan P (2013) Generalized Newtonian fluid simulations. In: *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*, 2013 Fourth National Conference on. IEEE, pp 1–4
87. Tubbs KR, Tsai FTC (2011) GPU accelerated lattice Boltzmann model for shallow water flow and mass transport. *Int J Numer Methods Eng* 86(3):316–334
88. Valero-Lara P (2017) Reducing memory requirements for large size LBM simulations on GPUs. *Concurr Comput Pract Exp* 29(24):e4221. <https://doi.org/10.1002/cpe.4221>
89. Wang C, Zhang Q, Kong F (2013) Simulation of free-surface flow using a boundless grid. *Sci China Inf Sci* 56(3):1–10
90. Wang Z, Zhao Y, Sawchuck AP, Dalsing MC, Yu H (2015) GPU acceleration of volumetric lattice Boltzmann method for patient-specific computational hemodynamics. *Comput Fluids* 115:192–200. <https://doi.org/10.1016/j.compfluid.2015.04.004>
91. Wei X, Zhao Y, Fan Z, Li W, Yoakum-Stover S, Kaufman A (2003) Blowing in the wind. In: *Proceedings of the 2003 ACM SIGGRAPH/eurographics symposium on computer animation*. Eurographics Association, pp 75–85
92. Wellein G, Zeiser T, Hager G, Donath S (2006) On the single processor performance of simple lattice Boltzmann kernels. *Comput Fluids* 35(8):910–919
93. Xian W, Takayuki A (2011) Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Comput* 37(9):521–535. <https://doi.org/10.1016/j.parco.2011.02.007>
94. Xiong Q, Li B, Xu J, Fang X, Wang X, Wang L, He X, Ge W (2012) Efficient parallel implementation of the lattice Boltzmann method on large clusters of graphic processing units. *Chin Sci Bull* 57(7):707–715
95. Xu A, Shi L, Zhao T (2017) Accelerated lattice Boltzmann simulation using GPU and OpenACC with data management. *Int J Heat Mass Transf* 109:577–588
96. Ye Y, Li K (2013) Entropic lattice Boltzmann method based high reynolds number flow simulation using CUDA on GPU. *Comput Fluids* 88:241–249
97. Zahran M (2017) Heterogeneous computing: here to stay. *Commun ACM* 60(3):42–45
98. Zhang J, Yuasa S, Fukuma S, Mori SI (2016) A real-time GPU-based coupled fluid-structure simulation with haptic interaction. In: *Computer and Information Science (ICIS)*, 2016 IEEE/ACIS 15th International Conference on. IEEE, pp 1–6
99. Zhao Y (2008) Lattice Boltzmann based PDE solver on the GPU. *Vis Comput* 24(5):323–333
100. Zhao Y, Han Y, Fan Z, Qiu F, Kuo YC, Kaufman AE, Mueller K (2007) Visual simulation of heat shimmering and mirage. *IEEE Trans Vis Comput Graph* 13(1):179–189
101. Zhao Y, Qiu F, Fan Z, Kaufman A (2007) Flow simulation with locally-refined LBM. In: *Proceedings of the 2007 symposium on interactive 3D graphics and games*. ACM, pp. 181–188
102. Zhao Y, Wang L, Qiu F, Kaufman A, Mueller K (2006) Melting and flowing in multiphase environment. *Comput Graph (Pergamon)* 30(4):519–528
103. Zhu H, Liu X, Liu Y, Wu E (2006) Simulation of miscible binary mixtures based on lattice Boltzmann method. *Comput Anim Virtual Worlds* 17(3–4):403–410