Received: 26 August 2022 | Revised: 19 January 2023 | Accepted: 7 February 2023

DOI: 10.1002/fld.5185

RESEARCH ARTICLE

WILEY

A graphic processing unit implementation for the moment representation of the lattice Boltzmann method

Marco A. Ferrari[1] | **Waine B. de Oliveira Jr.**[1] | **Alan Lugarini**[1] | **Admilson T. Franco**[1] | **Luiz A. Hegele Jr.**[2]

[1]Research Center for Rheology and Non-Newtonian Fluids (CERNN), Postgraduate Program in Mechanical and Materials Engineering, Federal University of Technology – Paraná (UTFPR), Curitiba, Brazil

[2]Department of Petroleum Engineering, Santa Catarina State University – UDESC, Balneário Camboriú, Brazil

Correspondence
Marco A. Ferrari, Research Center for Rheology and Non-Newtonian Fluids (CERNN), Postgraduate Program in Mechanical and Materials Engineering, Federal University of Technology – Paraná (UTFPR), Curitiba, PR 81280-340, Brazil.
Email: marcoferrari@alunos.utfpr.edu.br

Funding information
Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Grant/Award Number: 001; Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina, Grant/Award Number: 2021TR789

Abstract

We implement the moment representation of the lattice Boltzmann method in a graphic processing unit (GPU) environment to study the computational performance of three-dimensional fluid flows. The moment representation of the lattice Boltzmann method (MRLBM) uses up to second-order moments of the particle distribution function to regularize and reconstruct it. By adopting this method, there is a reduction in memory usage and global memory transfer and, therefore, increased performance. The results show an increase of up to 40% in speed compared to population schemes. The validation results show there was no loss of accuracy when using single precision for turbulent flows. The adoption of single-precision also enables higher processing speed for GPUs with low double-precision capability and reduces the memory footprint, which can further increase performance. The implementation here presented provides a fast and straightforward environment for fluid simulation by combining single-precision and the MRLBM.

KEYWORDS

GPU, lattice Boltzmann method, moment representation, parallel computing, single-precision

1 | INTRODUCTION

The lattice Boltzmann method (LBM) has become a popular technique for the computational solution of fluid flows.[1–9] The LBM is based on a mesoscopic approach to the continuum. The matter is modeled as a collection of particles that collide and move in a discrete velocity set according to a particle distribution function (populations).[1–4] Moments of the particle distribution function in the velocity space are then used to reconstruct macroscopic values, such as fluid density and velocity. Since most operations in LBM are local (typically, spatial derivatives are not present), this method performs very efficiently in parallel implementations on specialized hardware, such as graphics processing units (GPU).

In multi-core architectures, GPUs mainly,[10–22] a vast body of work is dedicated to improving the computational efficiency of the method, motivated by the need to solve demanding applications. Initially, there are two alternatives to implement the basic LBM algorithm: The first option is to use two sets of populations (denominated AB scheme, about one set named A and another set B). In this implementation, the populations are loaded from set A, where perform the collision step and then stored in set B. At the end of each time step, the memory pointers of A and B are swapped.[23] The AB scheme requires the allocation of 2×19 scalars (in the classic D3Q19 velocity set, for example) at every grid node.

An alternative to this memory-demanding scheme is the AA pattern, which uses one set of populations,[13,17,23–27] and therefore only the array A of populations is used. However, some streaming schemes need to be applied to guarantee that the populations are not overwritten when computing in a parallel environment. The first, presented by Bailey et al.,[23] uses two different patterns, where one pattern for even timesteps performs collision within the same lattice. While the second pattern is responsible for the even timesteps executing the streaming (by pulling the neighboring populations), followed by collision and streaming (by pushing) to original directions, assuring that only the populations being read are the ones being overwritten.

A substitute to AA-Pattern is the Esoteric scheme,[14,25] which pushes the populations that have a positive direction of the lattice itself, while the remaining populations, with a negative direction, are pulled from the neighboring lattices. The collision is then performed, and the populations are streamed to memory addresses that were previously read. Consequently, by the end of the time step, the populations are stored in an inverted position (e.g., index 1 is stored in index 2) and, therefore, need to be corrected by a pointer swap. Other streaming patterns that use one set of populations can be found in the literature: swift and swap streaming (SSS)[26] and compact streaming.[27]

The streaming patterns with one population set are preferred for large-scale computations, where the demand for refined domains is a priority while presenting a superior processing speed compared to other methods when CPUs were utilized.[28] The AB pattern could be preferred for GPUs in some cases where the domain size is not significant, but rather the number of timesteps is particularly large since it can exhibit a slight increase in speed[14,15,25] when compared to single-population streaming patterns. Nevertheless, when processing speed is not the priority but domain size, a single set population scheme should be used for GPU.

A third approach has been gaining attention recently: the moment representation of the LBM (MRLBM).[13,21,29] The MRLBM approach consists of regularizing and reconstructing the populations from their velocity moments, which reduces memory allocation by a significant factor,[13,21] since it is only necessary to store 10 moments in this approach. Hence, the memory consumption will be lower when compared to an AA pattern if the precision is unchanged.

Recent works utilized MRLBM and demonstrated noteworthy improvements in memory usage.[21] Compared to other methods, hybrid recursive regularised[30,31] or commulant,[32] the moment representation does not compute the high-order terms, leading to a more straightforward implementation, but at the cost of compressibility issues at high Reynolds numbers. Even though only the moments are stored in global memory, it is still necessary to reconstruct the populations to perform the streaming operation. Therefore, creating a buffer to store these variables between neighboring nodes, even temporarily, is necessary. Only a few works that used the MRLBM can be found in the current literature. Vardhan et al.[21] have opted to divide the domain into layers in the z-direction, and a moving buffer was responsible for storing and recomputing the moments when all streaming populations were obtained. A valid strategy when it is possible to restrict the order in which the lattices were calculated, as was the case in their CPU-based implementation. The other publication that used the moment representation of the LBM encountered in the literature is the work of Gounley et al.[13] Their work expanded the GPU-based implementation, but only for two-dimensional flows. In their approach, they divided the domain into tiles for each x coordinate and used a circular array shifting to reduce memory usage further. They reported a gain of 32% in performance when compared to the AB scheme for the two-dimensional case. However, if one uses the method in engineering problems, it is fundamental that it is GPU-implemented for three-dimensional problems.

The primary motivation behind this work is to incorporate the MRLBM in three dimensions for a GPU environment. The MRLBM reduces the memory footprint compared to other methods and, consequently, has the potential to allow larger domain sizes or the resolution of engineering simulations. With the MRLBM, the number of stored scalar reduces from 19 (in the classic D3Q19 velocity set with the AA scheme[23]) to 10 (0th–2nd order moments). The reduction of stored scalars decreases the frequency of reads and writes in the global memory. Since the GPUs are bandwidth limited, this characteristic should play a significant role in the computational performance. In this work, we derive the MRLBM with body force term, as other applications can be coupled with the LBM through it (for example, the immersed boundary method,[20] natural convection based on Boussinesq approximation[33]).

This work is organized as follows: in Section 2, we derive the theory of the MR-LBM with a body force term. Section 3 shows the implementation strategy for GPU computation. Section 4 brings the validation cases and the results for single versus double precision regarding accuracy in turbulent flows. Finally, Section 5 assessed the performance of the proposed numerical method in GPUs.

2 | NUMERICAL METHOD

The evolution equation for the discrete particle distribution function $f_i(\mathbf{r}, t)$, or the lattice Boltzmann equation, with the BGK collision operator, can be written as[1–4,6]:

$$f_i\left(\mathbf{r}+\mathbf{c}_i\Delta t, t+\Delta t\right)=f_i(\mathbf{r}, t)+\omega\Delta t\left(f_i^{(eq)}(\mathbf{r}, t)-f_i(\mathbf{r}, t)\right)+\left(1-\frac{\omega\Delta t}{2}\right)F_i(\mathbf{r}, t)\Delta t \tag{1}$$

where $\mathbf{r}$, $\mathbf{c}_i$, and t are dimensionless space, mesoscopic velocity, and time coordinates. The relaxation frequency is denoted by ω. The tensorial Hermite polynomials, up to second-order, are written as $\mathcal{H}_i^{(0)}=1$, $\mathcal{H}_{\alpha,i}^{(1)}=c_{i\alpha}$, and $\mathcal{H}_{\alpha\beta,i}^{(2)}=c_{i\alpha}c_{i\beta}-\delta_{\alpha\beta}/a_s^2$, where $\delta_{\alpha\beta}$ is the Kronecker delta and the subscripts α and β refer to Einstein summation notation, a_s is a scaling factor, which is $\sqrt{3}$. The discrete force term F_i is[34]:

$$F_i=w_i\left(a_s^2F_\alpha c_{i\alpha}+a_s^4F_\alpha u_\beta\mathcal{H}_{\alpha\beta,i}^{(2)}\right) \tag{2}$$

where w_i are the quadrature weights that depend on the direction i. We may also write $F_\alpha u_\beta$ as a sum of its symmetric and anti-symmetric counterparts since it contracts with a symmetric tensor $\mathcal{H}_{\alpha\beta,i}^{(2)}$:

$$F_\alpha u_\beta\mathcal{H}_{\alpha\beta,i}^{(2)}=\left[\frac{1}{2}\left(F_\alpha u_\beta+F_\beta u_\alpha\right)+\frac{1}{2}\left(F_\alpha u_\beta-F_\beta u_\alpha\right)\right]\mathcal{H}_{\alpha\beta,i}^{(2)}=\frac{1}{2}\left(F_\alpha u_\beta+F_\beta u_\alpha\right)\mathcal{H}_{\alpha\beta,i}^{(2)} \tag{3}$$

Consequently:

$$F_i=w_i\left[a_s^2F_\alpha c_{i\alpha}+\frac{1}{2}a_s^4\left(F_\alpha u_\beta+F_\beta u_\alpha\right)\mathcal{H}_{\alpha\beta,i}^{(2)}\right] \tag{4}$$

The function form of the equilibrium function $f_i^{(eq)}$ is taken as a second-order velocity expansion in the Hermite polynomials:

$$f_i^{(eq)}(\mathbf{r}, t)=\rho w_i\left(1+a_s^2u_\alpha c_{i\alpha}+\frac{1}{2}a_s^2u_\alpha u_\beta\mathcal{H}_{\alpha\beta,i}^{(2)}\right) \tag{5}$$

The velocity moments of f_i can be used to recover the macroscopic quantities such as density (ρ) and momentum flux, and the second-order moments ($m_{\alpha\beta}$), which are related to the stress tensor. They are[21]:

$$\rho(\mathbf{r}, t)=\sum_i f_i(\mathbf{r}, t) \tag{6}$$

$$\rho(\mathbf{r}, t)\overline{u}_\alpha(\mathbf{r}, t)=\sum_i f_i(\mathbf{r}, t)c_{i\alpha} \tag{7}$$

$$\rho(\mathbf{r}, t)m_{\alpha\beta}^{(2)}(\mathbf{r}, t)=\sum_i f_i(\mathbf{r}, t)\mathcal{H}_{\alpha\beta,i}^{(2)} \tag{8}$$

It should be noted that the velocity $\overline{u}_\alpha$ is not the hydrodynamic velocity but rather:

$$u_\alpha=\overline{u}_\alpha+\frac{1}{2\rho}F_\alpha \tag{9}$$

which is used in the equilibrium function. We regularize f_i, $f_i(\mathbf{r}, t)\rightarrow\widehat{f}(\mathbf{r}, t)$ up to second-order moments in the standard way[7]:

$$\widehat{f}(\mathbf{r}, t)=\rho w_i\left(1+a_s^2u_\alpha c_{i\alpha}+\frac{1}{2}a_s^4m_{\alpha\beta}^{(2)}\mathcal{H}_{\alpha\beta,i}^{(2)}\right) \tag{10}$$

Now, Equation (1) is rewritten in a short form as:

$$f_i^*=(1-\omega)\widehat{f}+\omega f_i^{(eq)}+\left(1-\frac{\omega}{2}\right)F_i \tag{11}$$

Where f^* is the post-collision population. The next step is to apply the moments (6)–(8) to the equations above, rendering the post-collision moments:

$$\rho^* = \sum_i f_i^*(\mathbf{r}, t) \tag{12}$$

$$\rho^* u_\alpha^* = \sum_i f_i^*(\mathbf{r}, t) c_{i\alpha} = (1-\omega)\rho u_\alpha + \omega \rho u_\alpha + \left(1 - \frac{\omega}{2}\right) F_\alpha \tag{13}$$

$$\rho^* m_{\alpha\beta}^{*(2)} = (1-\omega)\rho^* m_{\alpha\beta}^{(2)} + \omega \rho u_\alpha u_\beta + \left(1 - \frac{\omega}{2}\right)\left(F_\alpha u_\beta + F_\beta u_\alpha\right) \tag{14}$$

The collision is computed using Equations (12)–(14), comprising 10 operations on macroscopic values instead of 19 on mesoscopic variables. The return to the Navier–Stokes equations is obtained through the relation between ω and the fluid viscosity ν[2,6]:

$$\omega = \left(\frac{\nu}{c_s^2} + \frac{1}{2}\right)^{-1} \tag{15}$$

It is important to remember that the streaming step still needs to be done using the populations and not using the moments. Therefore the implementation strategy is crucial for this step, which is explained in the next session.

3 | GPU IMPLEMENTATION

We divide the fluid domain into blocks of size $B_x \times B_y \times B_z$ lattices each, resulting in $N_x/B_x \times N_y/B_y \times N_z/B_z$ total blocks, as shown in Figure 1. The blocks correspond to a CUDA block, and each thread will correspond to a lattice, where collision and streaming operations are performed. However, to avoid storing the streaming populations, it would be necessary to read the moments of the neighboring blocks and reconstruct the incoming populations, which would require excessive data transfer between blocks, limited by the machine's bandwidth. An alternative is to reconstruct the populations within the same block a single time and store those in the shared memory, therefore reducing the memory transfer between the GPU and the global memory, which now only needs to transfer the populations from lattices that are not in the same block.

However, the lattices in the blocks' borders will still require obtaining the population information from adjacent blocks. Therefore, it would be necessary to read all moments and reconstruct the streamed populations, which increases the number of memory reads, hurting the performance. In order to counter this problem, buffers zones responsible for transferring streamed populations between blocks are created by surfaces around the blocks, as illustrated in Figure 1. Similarly to the strategy adopted by Vardhan et al.[21] and Gounly et al.,[13] we extended it to a three-dimensions and without using the circular array-shifting strategy. The same strategy could be theoretically applied for a multi-GPU implementation (not performed in this work), with the convenience that the interfaces between the blocks within the GPU will coincide with the interfaces between two GPUs.

After initializing the moments on blocks and interface nodes, the main loop starts by loading the post-collision moments and converting them to post-collision populations through Equation (10). These populations are then stored in a shared memory array, allowing the streaming process to be performed within the block without transferring data from the global memory. The populations are pulled from the interface surfaces arrays for the lattices in the block surface. Next, if present, the boundary conditions are applied using the temporary populations of each block. Equations (6)–(8) evaluate the velocity moments, and the collision operation is performed through Equations (13) and (14). Finally, the post-collision moments are stored in the global memory, and for the interface lattices, the populations are recalculated and stored in the transfer surface array. Each time step, all these activities are performed in a single kernel launch.

The memory layout adopted for the moments' array is the structure of array of structures (SoAoS),[35] in which the moments coalesce together in the same block instead of the entire domain, see Figure 2. On the other hand, the streaming population arrays can only have one type of coalesced access, either read or write, since they move toward different grid lattices. Lehmann et al.[36] pointed out that having a coalesced write and misaligned read is preferable, which is what we adopt. Therefore, the post-collision populations for the neighboring lattices are stored together instead of grouped by the streaming target. The SoAoS memory layout is also used for the interface surfaces. The total memory usage for each block can be determined as follows:

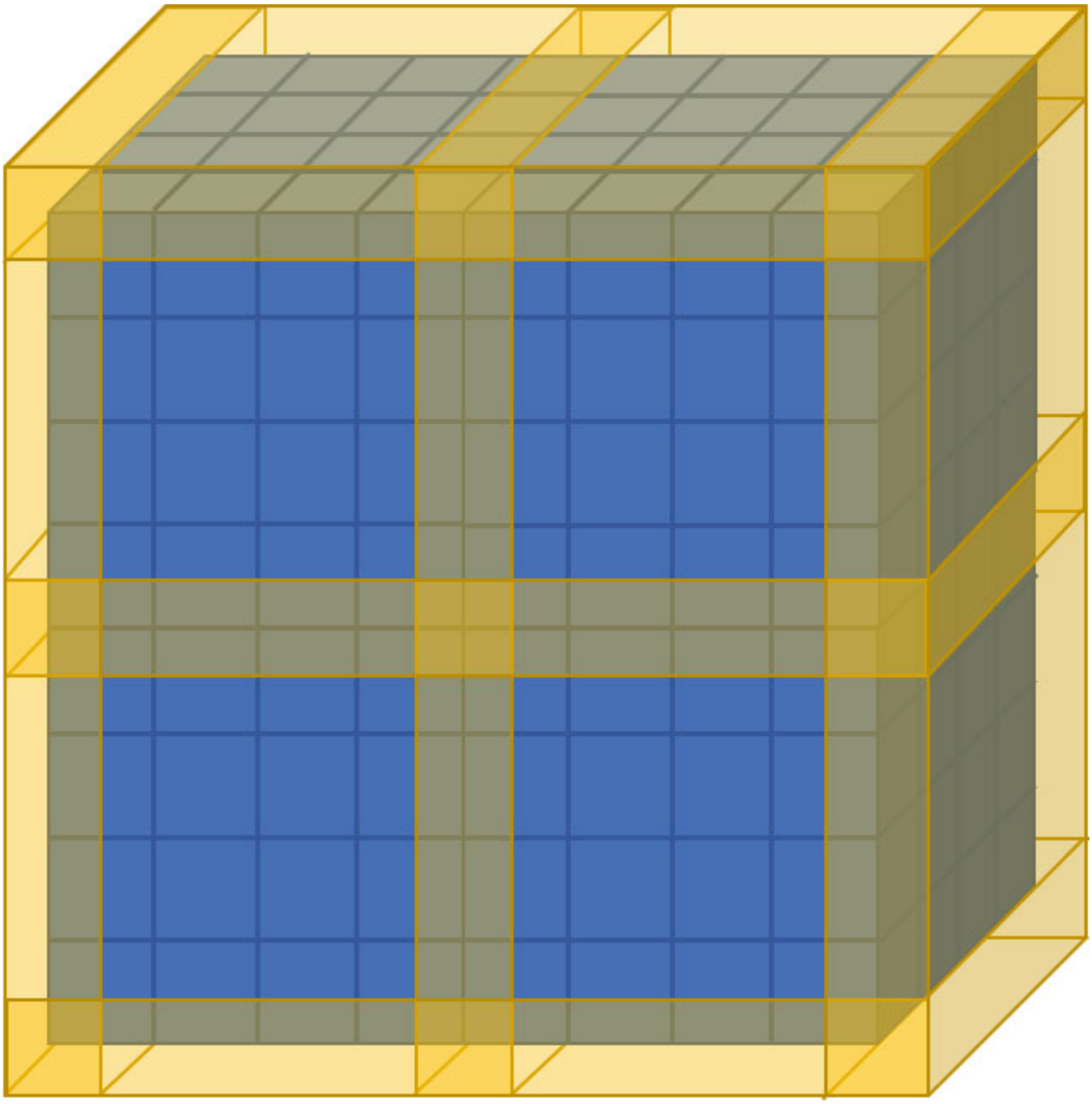

FIGURE 1 Representation of four blocks with size $4 \times 4 \times 4$, with lattices (blue) and interface surfaces (yellow). [Colour figure can be viewed at wileyonlinelibrary.com]

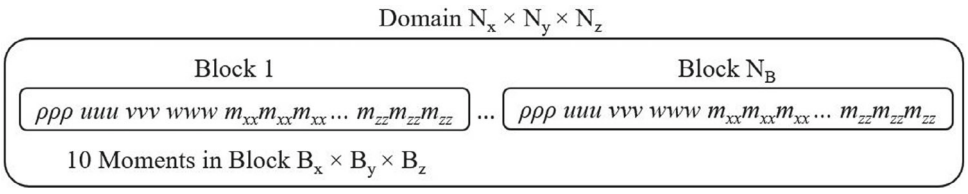

FIGURE 2 Representation of the structure of array of structures (SoAoS) array structure for the moments.

$$M = \frac{10\left(B_xB_yB_z\right) + 2\left(B_xB_y + B_xB_z + B_yB_z\right)Q_f}{B_xB_yB_z} \tag{16}$$

where Q_f is the number of transfer populations on each lattice face (10), and B is the block size. In order to avoid race conditions, two sets of transfer layers are used, hence factor 2, which is similar to the AB pattern. However, it is possible to implement a strategy similar to AA-pattern to eliminate it. For the current implementation, a block size of $4 \times 4 \times 4$ lattices increases memory usage from 19 to 25 scalars on the average per lattice, considering, for example, the D3Q19 velocity set. Nevertheless, when the block size changes to $8 \times 8 \times 8$ lattices, it is used an average of 17.5 scalars per lattice, a 7% reduction in the memory footprint. However, if an additional process requires knowing the macroscopic field, for example, when immersed boundary method is added to the model, the memory reduction further increases to 24% since the macroscopic values were already determined and stored in the global memory. Further increasing the block size would cause the average lattice size to lean toward 10 scalars per lattice on average, but the available shared memory of the GPU limits this change. Since populations of the boundary lattices in the block will only correspond to a small part of the total memory usage. The number of reads and writes from the global memory for each block is defined by the numerator of Equation (16); due to each moment, and boundary population is only required to be read and written once.

4 | VALIDATIONS

An essential discussion currently in vogue is the use of smaller floating-point variables to what is common practice in scientific simulations, that is, using 32-bit (single) instead of 64-bit (double) precision[11,22,36,37] while maintaining good numerical results. Recently,[36] even the adoption of mixed 32/16-bit precision showed promising results regarding the accuracy of the numerical method, with speed-ups of a factor of 4 when compared to FP64, since the memory bandwidth is a crucial bottleneck in GPUs.[13,21] Thus, by combining the 32-bit floating-point and the moment representation, a significant gain in speed and memory performance is expected. However, it is necessary to compare the results between FP32 and FP64 in some benchmark cases to guarantee that reduced precision will still be accurate. Two turbulent flow cases are used to assess the method's accuracy: Taylor–Green vortex and Turbulent channel flow.

4.1 | Taylor–Green Vortex

The simulation of the Taylor–Green vortex at the Reynolds number equal to $\mathrm{Re} = \rho UL/\mu = 1600$ is performed to verify the numerical code. The Taylor–Green vortex case is significant because it begins as laminar and progresses toward turbulence with smaller-scale vortices until viscous dissipation transforms all kinetic energy. The initial velocity field is initialized as[38]:

$$\begin{aligned} p &= p_0 + \frac{p_0U^2}{16}[\cos(2x/L) + \cos(2y/L)][2 + \cos(2z/L)]; \\ u &= U\sin(x/L)\cos(y/L)\cos(z/L); \\ v &= -U\cos(x/L)\sin(y/L)\cos(z/L); \\ w &= 0; \end{aligned} \tag{17}$$

where $U = 0.05$ and $L = N/(2\pi)$. The populations are initially calculated as the sum of the equilibrium and non-equilibrium populations and then posteriorly used to determine the initial field of the second-order moments. The kinetic energy dissipation rate is used to evaluate this case, and it is obtained through[39]:

$$\varepsilon U^2 = \frac{dE_k}{dt} = -\frac{1}{\rho_0 N^3}\frac{d}{dt}\left(\sum_{\text{lattices}}\frac{\rho u_i u_i}{2}\right) \tag{18}$$

The results, Figure 3, are compared with the numerical data using the spectral method obtained by Wang et al.[38] and the RLBM from our previous work.[9,22] For this case, the simulations ran with $N = 128$.

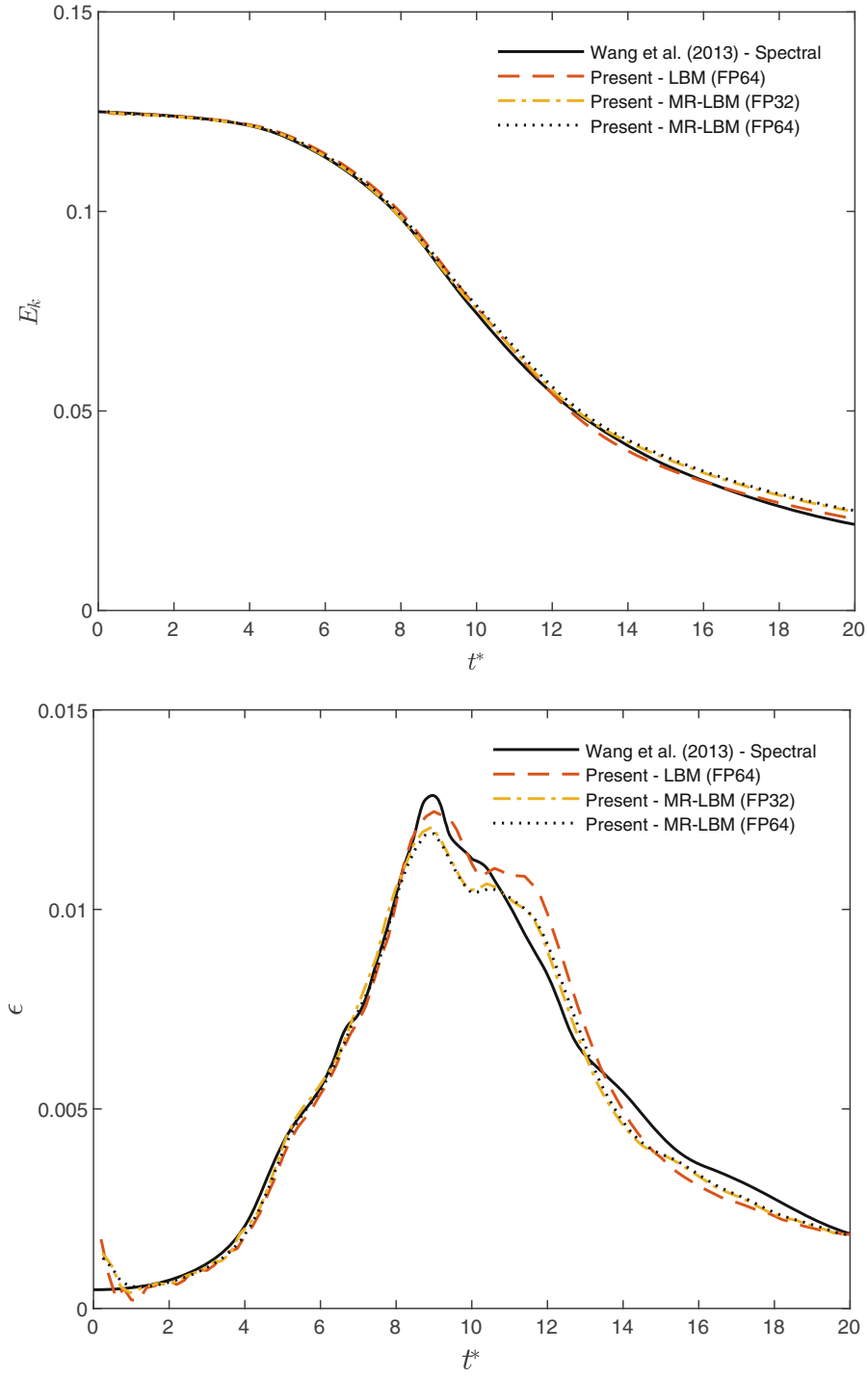

FIGURE 3 Total kinetic energy and dissipation rate for the Taylor–Green vortex at Re = 1600. [Colour figure can be viewed at wileyonlinelibrary.com]

The present results satisfactorily match the spectral data of Wang et al.,[38] considering the difference in the mesh utilized $N = 128$ versus 512. The dissipation rate increases at the same rate as the spectral reference, reaching its peak around $t^* = 9$, corresponding to the point with the highest local vorticity magnitude, Figure 4. After that, the dissipation rate decreases as the fluid settles, which can be seen in the total kinetic energy evolution, where the difference in the dissipation rate causes the total kinetic energy to diverge by 15% after $t^* = 20$. Regarding the use of FP32 versus FP64, there is not much difference in the model evolution: the difference in the total kinetic energy is less than 1% between the two precisions at the end of the simulation.

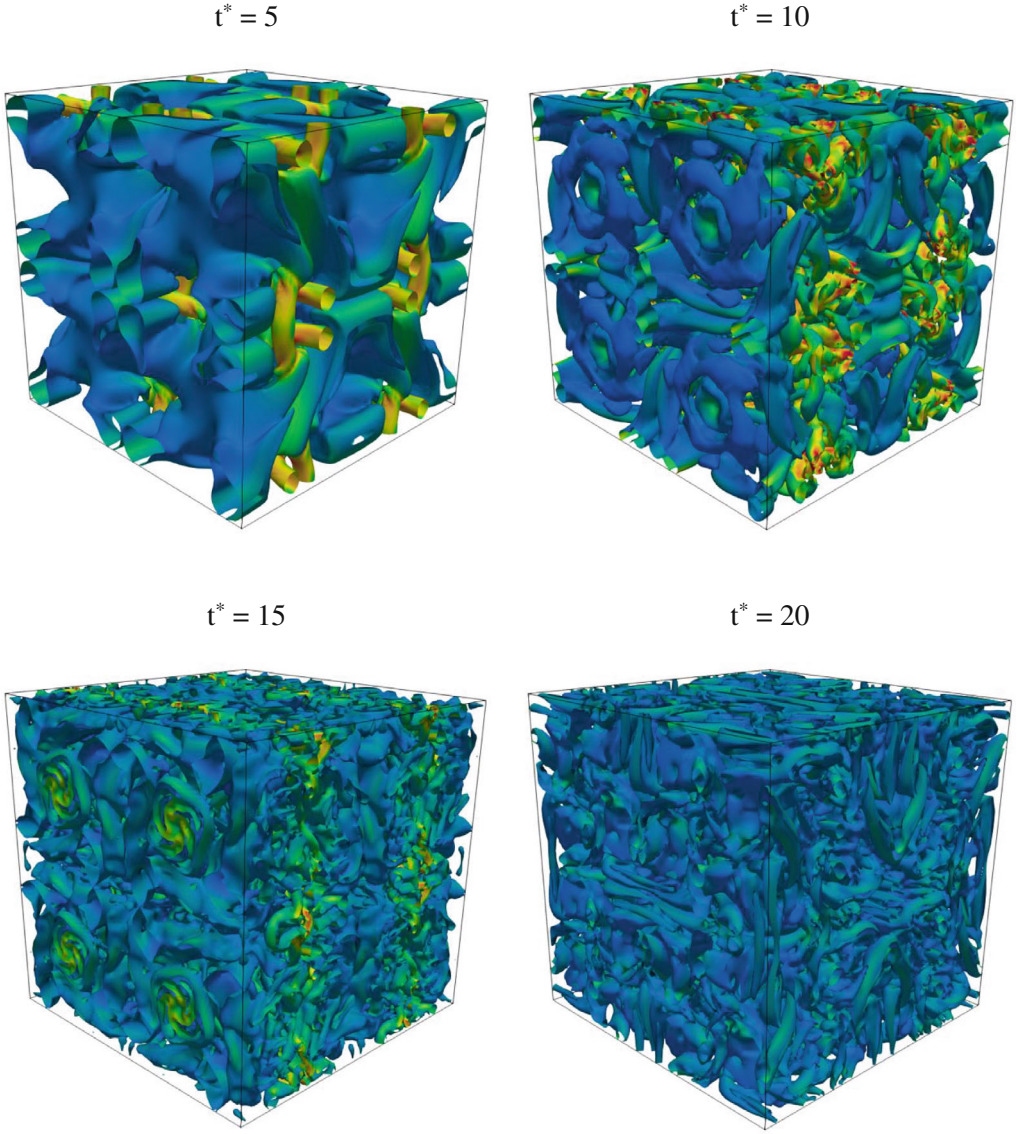

FIGURE 4 Isocontours of Q-criterion with vorticity magnitude. [Colour figure can be viewed at wileyonlinelibrary.com]

4.2 | Turbulent channel flow

Here we investigate the turbulent flow in a parallel plate channel. We compare with the results of Kim et al.[40] for the case $\mathrm{Re}_\tau = u_\tau \delta/\nu = 180$. The simulation is performed in a domain with $208 \times 208 \times 1664$ lattices, with a friction velocity of 0.00579 (Mach number of 0.01) and the channel center max velocity expected to reach ~0.09 (Mach number of 0.15). The single relaxation time is adopted $\tau = 0.5100$ for this case. The domain is initialized with a perturbation of 10% in the velocity field and run for 30 ETT (eddy turn-over time) to develop the flow fully. The statistics are collected for the next 50 ETT at ~20 times per ETT. The boundary condition applied to the walls is the Half-Way Bounce Back scheme,[41] which was checked in the laminar case and returned the L_2 error of order 2, corresponding to the value encountered in the literature. No turbulent wall model for this case is necessary since the model behaves similarly to a DNS with this mesh refinement. The mean velocity profile is compared with the literature data[40] and the law of the wall, shown in Figure 5. The result agrees with both, obeying the linear profile near the wall and transitioning to the logarithmic profile. The second-order statistics, shown in Figure 6, also agree with the results presented in the literature.

The difference is insignificant in the numerical results when using FP32 instead of FP64, as shown in Figures 5 and 6. We also show the instantaneous velocity and vorticity fields in Figure 7. The results are similar to those of Nathen et al.[42] in these fields, and no unphysical oscillations were found for $\mathrm{Re}_\tau = 180$.

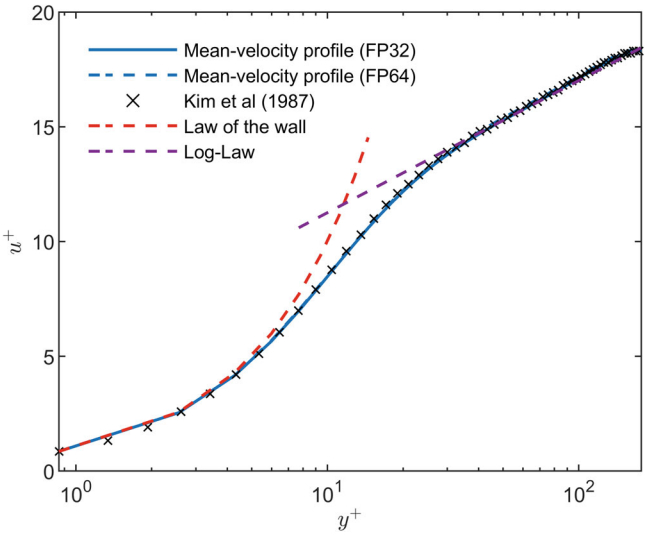

FIGURE 5 Mean velocity profile and comparison with the results of Kim et al.[40] and the law of the wall. [Colour figure can be viewed at wileyonlinelibrary.com]

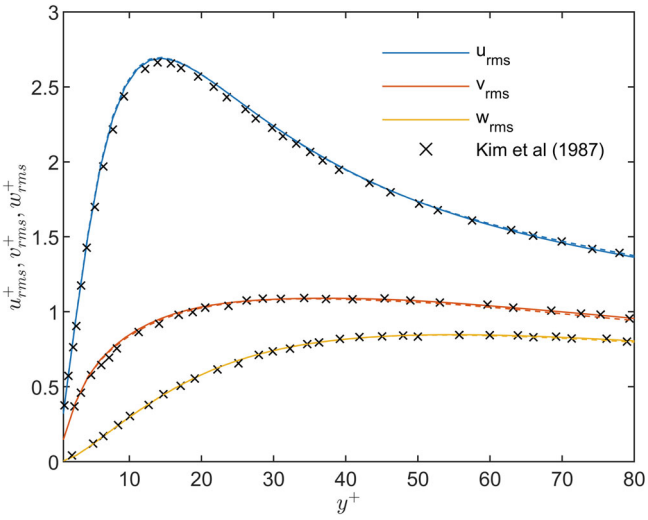

FIGURE 6 Root-mean-square velocity fluctuations profile and comparison with the results of Kim et al.[40] (–) FP32, (--) FP64. [Colour figure can be viewed at wileyonlinelibrary.com]

Overall, using FP32 and FP64 for the cases analyzed in this work did not cause the final solution to alter significantly. The obtained result is similar to Lehmann et al.[36] Therefore, we conclude that using FP32 is not detrimental to the overall result.

5 | PERFORMANCE

The performance analysis was executed in selected NVIDIA® GPUs: A100, K80, RTX 3060, and GTX 1660. Their technical specifications are listed in Table 1. All code was compiled with NVCC v11.4.100 with O3 optimization using the Taylor–Green vortex case with a domain size of $256 \times 256 \times 256$*, 20,000-time steps, and D3Q19 velocity set. Data processing, reports, or saving processes were disabled as they impact performance measurements.[22] For A100 and K80, ECC was enabled, while the remaining ones do not have this feature. The results of the AB pattern were taken from previous works code.[9,22]

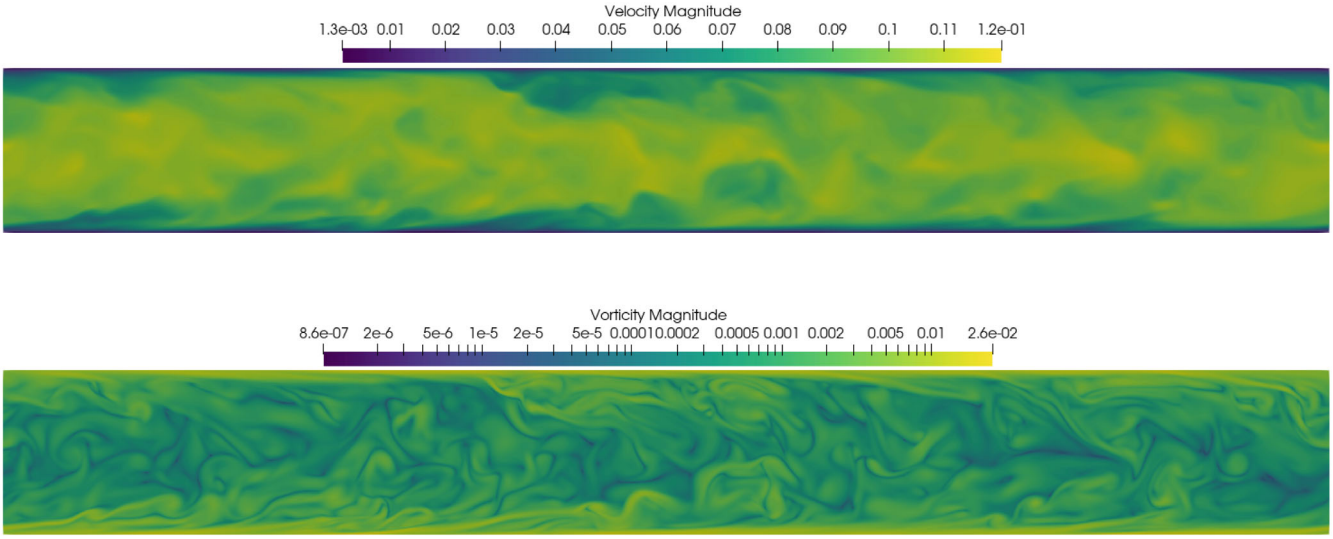

FIGURE 7 Instantaneous velocity and vorticity fields in *xy*-plane. [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Technical specifications of NVIDIA® GPUs.

GPU	A100	K80	RTX 3060	GTX 1660
CUDA cores	6912	2496	3584	1408
Streaming multiprocessor	108	13	28	22
Memory	40 GB HBM2e	12 GB GDDR5	12 GB GDDR6	6 GB GDDR5
Bandwidth	1555 GB/s	240.6 GB/s	360 GB/s	192 GB/s
L1 (per SM)/L2 Cache	196 KB/40 MB	16 KB/1.5 MB	128 KB/3 MB	64 KB/1.5 MB
CUDA Version	8.0	3.7	8.6	7.5
Architecture	Ampere (1:2)	Kepler (1:3)	Ampere (1:64)	Turing (1:32)

Abbreviation: GPU, graphic processing unit.

The roofline performance analysis[43] is performed using NVIDIA.[44] For this case, the adopted block size was $8 \times 8 \times 8$ for FP32 and $8 \times 8 \times 4$ for FP64 since those are ones of the most significant possible block sizes that keep the number of threads as a multiple of the warp size (32) to try to achieve a high warp occupancy.[45] The results are depicted in Figure 8.

Since there is little change in the compiled code structure among the different GPUs, all of them present a similar arithmetic intensity (AI), which for the case analyzed is ~4.7 flop/byte for FP32 and 2.1 flop/byte for FP64. For FP32, this level of AI causes the application to be limited by the memory bandwidth. By changing the precision to FP64, the non-Tesla lineup GPUs transition from being bandwidth limited to being computed-limited because the GPU cannot quickly process the data since it has a ratio of FP32/FP64 of either 1:32 or 1:64. However, this does not occur for the Tesla lineup GPU since those cards are designed to work with FP64 and, consequently, have a higher ratio FP32/FP64 performance (e.g., 1:2 for A100). As a result, these GPUs are still limited by the device's bandwidth. The Tesla A100 is a particular case due to its memory bus size (5120 bits), which could not be saturated in the execution of a single warp (the effective bandwidth is 1035 GB/s for FP32 and 1270 GB/s for FP64 instead of the nominal 1555 GB/s) and therefore exhibit a reduced performance compared to the theoretical maximum. Because of the higher bandwidth achieved for FP64, the performance is slightly above 50% of the FP32.

The presented roofline model helps to explain the performance of LBM in MLUPS (Mega lattices updates per second), Figure 9, representing the usual performance metric for LBM simulations.

By increasing the block size, there is a reduction in the average number of populations that need to be streamed from the block's frontier over the total number of lattices in the block. Consequently, there is a decrease in the time that the

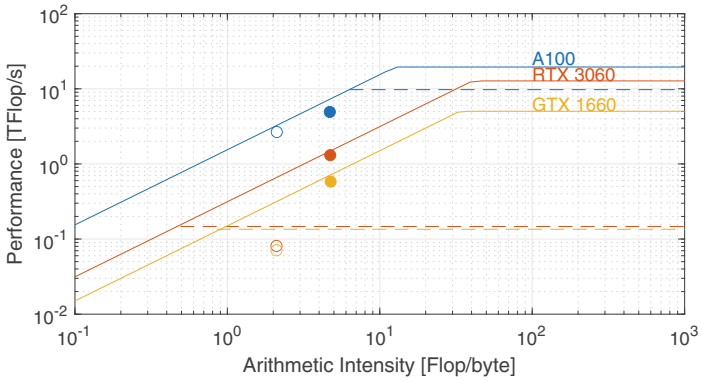

FIGURE 8 Roofline performance model of the moment representation of the lattice Boltzmann method (MR-LBM) in three selected graphic processing units (GPUs). Filled markers correspond to FP32, while emptied ones FP64. Dashed rooflines represent the FP64 compute limit. [Colour figure can be viewed at wileyonlinelibrary.com]

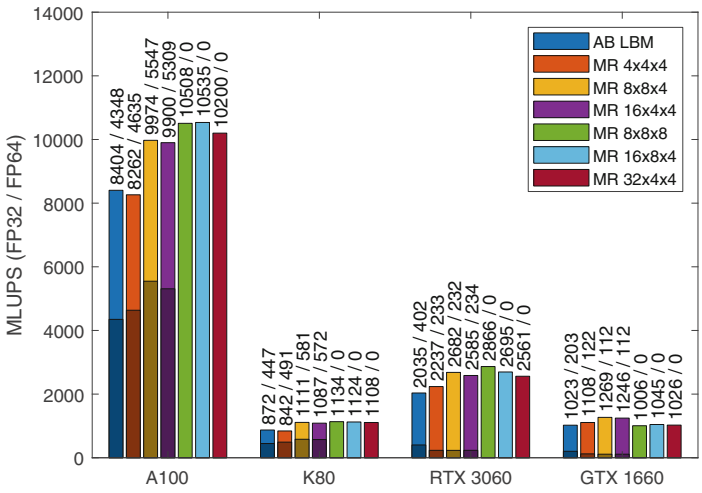

FIGURE 9 Performance of moment representation of the lattice Boltzmann method (MR-LBM) as a function of the block size in different graphic processing unit (GPU) architectures. [Colour figure can be viewed at wileyonlinelibrary.com]

simulation takes to be completed, as seen by comparing $4 \times 4 \times 4$ versus $8 \times 8 \times 4$ for all devices. Changing the format of the block from $8 \times 8 \times 4$ to $16 \times 4 \times 4$ did not have a big difference in performance. However, when going from $8 \times 8 \times 8$ to $32 \times 4 \times 4$, there is a performance drop of 10% and 18% for the RTX3060 and GTX 1660, respectively. The drop in performance was not perceived in the A100/K80. Despite the K80 having more bandwidth than the GTX 1660, the ECC impacts around 19% of the overall performance in the K80,[22] making both GPUs have similar apparent bandwidth and, thus, similar MLUPS.

Regarding the FP64, the A100 and K80 exhibit around half of the MLUPS of FP32, as these cards are still limited by bandwidth and not the computing power of the GPU. Meanwhile, in the RTX 3060 and GTX 1660, there was a 10-fold decrease in the application velocity, higher than the FP32/FP64 compute ratios of 1:64 and 1:32, respectively, which occurs because there is a surplus of compute power when using FP32, see Figure 8, that is now utilized.

Comparing the MR-LBM with the AB scheme (which is within a 5% difference from the results of Lehmann et al.[36] and Amati et al.[46] for the A100), all GPUs had an overall increase in the speed by using FP32, when employing a suitable block size, of about 20%–40%. The performance gain is inferior to the gain observed by Gounley et al.[13] in their CPU implementation. Meanwhile, for FP64, the RTX3060 and GTX 1660 decreased by around 40% in performance compared with the AB LBM. The difference in performance occurs because the AB LBM has a lower number of arithmetic operations than the MR-LBM, and therefore when limited by the computing power of the GPU, the process with lower arithmetic operations will have a higher MLUPS.

6 | CONCLUSION

In this paper, we developed and tested a GPU implementation of the MRLBM in three-dimensional flows and evaluated its accuracy and performance.

It was observed that the computational performance increases for larger blocks as long as the computation is limited by bandwidth because the amount of streaming populations relative to the lattices in the blocks reduces. The block format is also essential since slender blocks exhibit a higher ratio of surface to volume. In the case of compute-bound situations (devices with low ratio FP64:FP32 performance), the moment representation exhibited reduced performance concerning the AB scheme because it has a higher AI. Nonetheless, in the best-case scenario (FP32 with the largest block size), the MLBM had an MLUPS increase of up to 40% compared to the AB LBM scheme. Furthermore, this method can be used in industrial applications to help increase the resolution and, therefore, model smaller turbulence scales with higher computational speed than previously possible with the same hardware.

CRediT Statement

Marco A. Ferrari: Conceptualization; methodology; software; validation; formal analysis; investigation; writing—original draft; visualization. Waine B. de Oliveira Jr.: Conceptualization; methodology; software. Alan Lugarini: Resources; writing—review & editing; supervision; project administration. Admilson T. Franco: Writing—review & editing; supervision; project administration; funding acquisition. Luiz A. Hegele Jr.: Conceptualization; methodology; writing—review & editing; supervision; project administration.

ACKNOWLEDGMENTS

The authors acknowledge the financial support of the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES), Finance Code 001. L.A. Hegele Jr. would like to acknowldge FAPESC (Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina) through grant Number 2021TR789 for financial support. We thank Rodrigo S. Romanus for the discussions that helped elaborate this work. We also thank the anonymous reviewers for their helpful suggestions and comments.

CONFLICT OF INTEREST STATEMENT

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

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

CODE AVAILABILITY

The numerical code developed is available at https://github.com/CERNN/MR-LBM

ENDNOTE

*For the GTX 1660 AB LBM FP64 the results were obtained using $128 \times 128 \times 128$ domain due memory size.

ORCID

Marco A. Ferrari https://orcid.org/0000-0003-0652-2144
Waine B. de Oliveira Jr. https://orcid.org/0000-0001-5185-8814
Alan Lugarini https://orcid.org/0000-0002-5704-5280
Admilson T. Franco https://orcid.org/0000-0002-7977-6404
Luiz A. Hegele Jr. https://orcid.org/0000-0001-9329-5036

REFERENCES

1. He X, Luo L-S. Theory of the lattice Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation. *Phys Rev E.* 1997;56(6):6811-6817. doi:10.1103/PhysRevE.56.6811
2. Succi S. The lattice Boltzmann equation for fluid dynamics and beyond. Oxford University Press. 2001:304.

3. Latt J, Chopard B. Lattice Boltzmann method with regularized pre-collision distribution functions. *Math Comput Simul*. 2006;72(2–6):165-168. doi:10.1016/j.matcom.2006.05.017
4. Philippi PC, Hegele LA, dos Santos LOE, Surmas R. From the continuous to the lattice Boltzmann equation: the discretization problem and thermal models. *Phys Rev E*. 2006;73(5):056702. doi:10.1103/PhysRevE.73.056702
5. Latt J, Chopard B, Malaspinas O, Deville M, Michler A. Straight velocity boundaries in the lattice Boltzmann method. *Phys Rev E*. 2008;77(5):056703. doi:10.1103/PhysRevE.77.056703
6. Silva G, Semiao V. First-and second-order forcing expansions in a lattice Boltzmann method reproducing isothermal hydrodynamics in artificial compressibility form. *J Fluid Mech*. 2012;698:282-303. doi:10.1017/jfm.2012.83
7. Hegele LA, Scagliarini A, Sbragaglia M, et al. High-Reynolds-number turbulent cavity flow using the lattice Boltzmann method. *Phys Rev E*. 2018;98(4):043302. doi:10.1103/PhysRevE.98.043302
8. Geier M, Lenz S, Schönherr M, Krafczyk M. Under-resolved and large eddy simulations of a decaying Taylor–Green vortex with the cumulant lattice Boltzmann method. *Theor Comput Fluid Dyn*. 2021;35(2):169-208. doi:10.1007/s00162-020-00555-7
9. Ferrari MA, Lugarini A, Franco AT. Fully-resolved simulations of a sphere settling in an initially unstructured thixo-viscoplastic fluid. *J Nonnewton Fluid Mech*. 2021;294:104574. doi:10.1016/j.jnnfm.2021.104574
10. Xian W, Takayuki A. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Comput*. 2011;37(9):521-535. doi:10.1016/j.parco.2011.02.007
11. Obrecht C, Kuznik F, Tourancheau B, Roux J-JJ. Multi-GPU implementation of the lattice Boltzmann method. *Comput Math Appl*. 2013;65(2):252-261. doi:10.1016/j.camwa.2011.02.020
12. Ames J, Puleri DF, Balogh P, Gounley J, Draeger EW, Randles A. Multi-GPU immersed boundary method hemodynamics simulations. *J Comput Sci*. 2020;44:101153. doi:10.1016/j.jocs.2020.101153
13. Gounley J, Vardhan M, Draeger EW, Valero-Lara P, Moore SV, Randles A. Propagation pattern for moment representation of the lattice Boltzmann method. *IEEE Trans Parallel Distrib Syst*. 2022;33(3):642-653. doi:10.1109/TPDS.2021.3098456
14. Lehmann M. Esoteric pull and esoteric push: two simple in-place streaming schemes for the lattice Boltzmann method on GPUs. *Comput Secur*. 2022;10(6):92. doi:10.3390/computation10060092
15. Januszewski M, Kostur M. Sailfish: a flexible multi-GPU implementation of the lattice Boltzmann method. *Comput Phys Commun*. 2014;185(9):2350-2368. doi:10.1016/j.cpc.2014.04.018
16. Mawson MJ, Revell AJ. Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture nVidia GPUs. *Comput Phys Commun*. 2014;185(10):2566-2574. doi:10.1016/j.cpc.2014.06.003
17. Valero-Lara P, Igual FD, Prieto-Matías M, Pinelli A, Favier J. Accelerating fluid-solid simulations (lattice-Boltzmann and immersed-boundary) on heterogeneous architectures. *J Comput Sci*. 2015;10:249-261. doi:10.1016/j.jocs.2015.07.002
18. Valero-Lara P. Leveraging the performance of LBM-HPC for large sizes on GPUs using ghost cells. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence, Lecture Notes in Bioinformatics)*. Vol 10048. LNCS; 2016:417-430. doi:10.1007/978-3-319-49583-5_31
19. Ren F, Song B, Zhang Y, Hu H. A GPU-accelerated solver for turbulent flow and scalar transport based on the lattice Boltzmann method. *Comput Fluids*. 2018;173:29-36. doi:10.1016/j.compfluid.2018.03.079
20. Beny J, Latt J. Efficient LBM on GPUs for dense moving objects using immersed boundary condition; 2019;(1):1-14. http://arxiv.org/abs/1904.02108
21. Vardhan M, Gounley J, Hegele L, Draeger EW, Randles A. Moment representation in the lattice Boltzmann method on massively parallel hardware. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM; 2019:1-21. doi:10.1145/3295500.3356204
22. Oliveira WB, Lugarini A, Franco AT. Performance Analysis of the Lattice Boltzmann Method Implementation on GPU. *XL Ibero-Latin-American Congr Comput Methods Eng* (*CILAMCE 2019*). Published online 2019.
23. Bailey P, Myre J, SDC W, Lilja DJ, Saar MO. Accelerating lattice Boltzmann fluid flow simulations using graphics processors. *International Conference on Parallel Processing*. IEEE; 2009:550-557. doi:10.1109/ICPP.2009.38
24. Latt J, Coreixas C, Beny J. Cross-platform programming model for many-core lattice Boltzmann simulations. Tian F-B, ed. *PLoS One*. 2021;16(4):e0250306. doi:10.1371/journal.pone.0250306
25. Geier M, Schönherr M. Esoteric twist: an efficient in-place streaming algorithmus for the lattice Boltzmann method on massively parallel hardware. *Comput Secur*. 2017;5(4):19. doi:10.3390/computation5020019
26. Mohrhard M, Thäter G, Bludau J, Horvat B, Krause MJ. Auto-vectorization friendly parallel lattice Boltzmann streaming scheme for direct addressing. *Comput Fluids*. 2019;181:1-7. doi:10.1016/j.compfluid.2019.01.001
27. Perepelkina A, Levchenko V, Zakirov A. New compact streaming in LBM with ConeFold LRnLA algorithms. *Communications in Computer and Information Science*. Vol 1331. Springer International Publishing; 2020:50-62. doi:10.1007/978-3-030-64616-5_5
28. Wittmann M, Zeiser T, Hager G, Wellein G. Comparison of different propagation steps for lattice Boltzmann methods. *Comput Math Appl*. 2013;65(6):924-935. doi:10.1016/j.camwa.2012.05.002
29. Latt J. Hydrodynamic limit of lattice Boltzmann equations. 10.13097/archive-ouverte/unige:464
30. Jacob J, Malaspinas O, Sagaut P. A new hybrid recursive regularised Bhatnagar–Gross–Krook collision model for lattice Boltzmann method-based large eddy simulation. *J Turbul*. 2018;19(11–12):1051-1076. doi:10.1080/14685248.2018.1540879
31. Feng Y, Boivin P, Jacob J, Sagaut P. Hybrid recursive regularized lattice Boltzmann simulation of humid air with application to meteorological flows. *Phys Rev E*. 2019;100(2):023304. doi:10.1103/PhysRevE.100.023304

32. Geier M, Schönherr M, Pasquali A, Krafczyk M. The cumulant lattice Boltzmann equation in three dimensions: theory and validation. *Comput Math Appl.* 2015;70(4):507-547. doi:10.1016/j.camwa.2015.05.001
33. He X, Chen S, Doolen GD. A novel thermal model for the lattice Boltzmann method in incompressible limit. *J Comput Phys*. 1998;146(1):282-300. doi:10.1006/jcph.1998.6057
34. Shan X, Yuan X-F, Chen H. Kinetic theory representation of hydrodynamics: a way beyond the Navier–Stokes equation. *J Fluid Mech*. 2006;550(1):413. doi:10.1017/S0022112005008153
35. Shet AG, Siddharth K, Sorathiya SH, et al. On vectorization for lattice based simulations. *Int J Mod Phys C*. 2013;24(12):1340011. doi:10.1142/S0129183113400111
36. Lehmann M, Krause MJ, Amati G, Sega M, Harting J, Gekle S. On the accuracy and performance of the lattice Boltzmann method with 64-bit, 32-bit and novel 16-bit number formats; 2021. http://arxiv.org/abs/2112.08926
37. Kuznik F, Obrecht C, Rusaouen G, Roux JJ. LBM based flow simulation using GPU computing processor. *Comput Math Appl.* 2010;59(7):2380-2392. doi:10.1016/j.camwa.2009.08.052
38. Wang ZJ, Fidkowski K, Abgrall R, et al. High-order CFD methods: current status and perspective. *Int J Numer Methods Fluids*. 2013;72(8):811-845. doi:10.1002/fld.3767
39. Brachet ME, Meiron D, Orszag S, Nickel B, Morf R, Frisch U. The Taylor-Green vortex and fully developed turbulence. *J Stat Phys*. 1984;34(5–6):1049-1063. doi:10.1007/BF01009458
40. Kim J, Moin P, Moser R. Turbulence statistics in fully developed channel flow at low Reynolds number. *J Fluid Mech*. 1987;177:133-166. doi:10.1017/S0022112087000892
41. Ladd AJC. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *J Fluid Mech*. 1994;271(1994):285-309. doi:10.1017/S0022112094001771
42. Nathen P, Gaudlitz D, Krause MJ, Adams NA. On the stability and accuracy of the BGK, MRT and RLB Boltzmann schemes for the simulation of turbulent flows. *Commun Comput Phys*. 2018;23(3):846-876. doi:10.4208/cicp.OA-2016-0229
43. Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures. *Commun ACM*. 2009;52(4):65-76. doi:10.1145/1498765.1498785
44. NVIDIA. Nsight compute command line interface. *V202121*. 2021; (January). https://docs.nvidia.com/nsight-compute/NsightComputeCli/
45. NVIDIA. CUDA Toolkit Documentation; 2022. https://docs.nvidia.com/cuda/archive/11.6.0/
46. Amati G, Succi S, Fanelli P, Krastev VK, Falcucci G. Projecting LBM performance on exascale class architectures: a tentative outlook. *J Comput Sci*. 2021;55(September):101447. doi:10.1016/j.jocs.2021.101447

How to cite this article: Ferrari MA, de Oliveira WB, Lugarini A, Franco AT, Hegele LA. A graphic processing unit implementation for the moment representation of the lattice Boltzmann method. *Int J Numer Meth Fluids*. 2023;95(7):1076-1089. doi: 10.1002/fld.5185