The lattice Boltzmann method for turbulent channel flows using graphics processing units

V. Zecevic[1] M. P. Kirkpatrick[2] S. W. Armfield[3]

(Received 30 January 2011; revised 1 November 2011)

Abstract

We performed a direct numerical simulation of turbulent channel flow at Reynolds number 180 using the Lattice Boltzmann method. We used the single relaxation time collision operator. The code was executed using graphics processing units as a highly parallel high performance computing platform. Results are compared to published direct numerical simulation results. We avoid common drawbacks of the method, such as compressibility error and instability at higher Reynolds numbers, by using a sufficiently small Mach number and lattice spacing. We validate the Lattice Boltzmann method using the single relaxation time collision operator as an effective tool for continued research into fundamental turbulent flows. The method is less suitable for wall bounded turbulence since these flows benefit from an increased resolution near the wall while the standard Lattice Boltzmann method requires an isotropic, homogeneous lattice. This

http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/3951 gives this article, © Austral. Mathematical Soc. 2011. Published November 10, 2011. ISSN 1446-8735. (Print two pages per sheet of paper.)

work validates the method as well as being a guide to suitable parameter ranges and target flows.

Contents

1 Introduction

The lattice Boltzmann (LB) method has enjoyed a rise in popularity over recent years. When using the simple single relaxation time (SRT) collision operator, the method becomes an artificially compressible Navier–Stokes solver. The method is fully explicit, making it attractive for parallel computing, and there is no pressure velocity coupling, greatly reducing the computational effort compared to finite volume methods. The LB method is able to deal with

complex boundaries and is extensible to include multiple phases, species and thermal effects. A recent review was presented by Aiden and Clausen [1].

Graphics processing units (GPUs) have become increasingly powerful, offering theoretical performance well in excess of normal consumer grade processors. Their massively parallel architecture means that this power can only be used effectively on a subset of suitable problems. The explicit and local nature of the LB method means it responds very well to GPU computing. Bernaschi et al. [4] used the LB method for flow through complex geometries and report a three fold increase in speed using eight Nvidia GT200 series GPUs compared to 512 nodes of an IBM BlueGene cluster. Bailey et al. [2] and Toelke and Krafczyk [14] also reported high performance GPU implementations of the method.

Peng and colleagues [13] validate the LB method for decaying turbulence by comparing results to the spectral method. Kareem et al. [9] also verify the LB method for homogeneous, isotropic turbulence. Bespalko, Pollard and Udin [5] recently simulated turbulent channel flow using the LB method, finding general agreement but noting that the uniform grid spacing was a drawback.

2 The lattice Boltzmann method

The LB method tracks fictitious particle populations that move along a fixed isotropic lattice. Each time step consists of a collision between the particles, followed by the populations streaming to the next lattice site in their path. During the collision step, populations at a particular site are redistributed in accordance with some collision rule. Using the SRT collision model, this step is a relaxation toward local equilibrium. The time advancement is summarised as

$$f_i\left(\mathbf{x}+\mathbf{c}_i\cdot\Delta t, t+\Delta t\right) = (1-\omega)\, f_i\left(\mathbf{x}, t\right) + \omega f_i^{eq}\left(\mathbf{x}, t\right). \qquad (1)$$

Although Δt is always one, it is included so that equations maintain the correct dimensions. Each lattice is associated with a lattice tensor $\mathbf{c}_i$ that represents the velocity of the ith particle direction. Each direction is also assigned a weight W_i. The hydrodynamic fields are calculated as

$$\rho(\mathbf{x}, t) = \sum f_i(\mathbf{x}, t) \quad \text{and} \quad \mathbf{u}(\mathbf{x}, t) = \frac{1}{\rho(\mathbf{x}, t)} \sum f_i(\mathbf{x}, t)\, \mathbf{c}_i \,. \tag{2}$$

With the equilibrium distribution defined as

$$f_i^{eq}(\mathbf{x}, t) = \rho W_i \left[1 + \frac{\mathbf{c}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{c}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u} \cdot \mathbf{u}}{2c_s^2} \right], \tag{3}$$

the scheme asymptotically approaches the incompressible Navier–Stokes (NS) equations in the low Mach number limit, is second order accurate in space and has first order accuracy in Mach number, equivalent to it being first order in time.

The kinematic viscosity ν and pressure P are defined in terms of the relaxation factor ω and the speed of sound c_s:

$$\nu = c_s^2 \cdot \Delta t \left(\frac{1}{\omega} - \frac{1}{2} \right) \quad \text{and} \quad P = c_s^2 \rho \,. \tag{4}$$

In general, c_s depends on the lattice used; however, for all lattices used in this work $c_s = 1/\sqrt{3}$.

Rigorous proof of this NS equivalence and order of accuracy was given by Benzi, Succi and Vergassola [3], Wolf and Gladrow [15] and Junk, Klar and Luo [7]. These works also discussed isotropy requirements used to derive the D3Q15 and D3Q19 lattices used in the present work.

3 Problem specification

We simulate a fully resolved turbulent channel flow in a periodic domain. The domain is a rectangular prism with periodic boundaries in the streamwise x and

Table 1: Dimensions of lattices used.

	N_y	N_x	N_z	l_y	l_x	l_z	N_{tot}	y^+_{wall}
LBM - 92	92	288	176	2	2π	π	4.6×10^6	3.82
LBM - 112	112	352	176	2	2π	π	6.9×10^6	3.16
LBM - 132	132	416	212	2	2π	π	11.6×10^6	2.69
LBM - 152	152	480	246	2	2π	π	17.8×10^6	2.35
LBM - 92 - 4π	92	576	198	2	4π	$4/3\,\pi$	10.4×10^6	3.82
LBM - 112 - 4π	112	704	234	2	4π	$4/3\,\pi$	18.0×10^6	3.16
Moser et al. [12]	129	128	128	2	4π	$4/3\,\pi$	2.1×10^6	0.05

spanwise z directions and solid wall boundaries in the wall-normal y direction. The half height of the channel δ is equal to one. Results are obtained for streamwise domain lengths of 2π and 4π and spanwise widths of π and $4/3\pi$ and compared to the spectral results of Moser, Kim and Mansour [12]. Table 1 details the domain size and lattice resolutions used here, and those of Moser et al. Our implementation of the LB method requires the use of a uniform lattice, resulting in a coarser mesh at the wall, as compared to NS methods which commonly use nonuniform grids. The distance to the node closest to the wall is given in terms of wall units

$$y^+ = \frac{u_\tau y}{\nu}\,. \tag{5}$$

The uniform grids result in a slightly higher wall normal resolution in the centre of the channel and a much higher resolution in the streamwise and spanwise directions. The flow is driven by a body force equivalent to a pressure gradient.

The simplest solid wall boundary condition (BC) bounces particles back along their incoming path at solid walls. This 'bounce back' BC is only first order in space while the rest of the method is second order. We use the extrapolation BC introduced by Guo, Zheng and Shi [6] in order to maintain second order accuracy. Here, one layer of ghost nodes is maintained outside the fluid with the boundary falling anywhere in the region between the ghost node and

the fluid node. This BC has the capacity to deal with curved boundaries; however, we use a simplified version to deal with a boundary aligned with the Cartesian grid, and place the boundary exactly on the ghost node.

4 Implementation

Mattila et al. [10, 11] outlined many implementations of the LB method. We use a 'two step' method where the collision and streaming operation are performed separately and only one copy of lattice information is stored. This method is a factor of two slower than the 'two lattice' method where the collision and streaming are fused into one step, writing results to a second copy of the lattice to remove data dependence. Mattila et al. [10, 11] also present a more efficient 'swap' algorithm where the collision and streaming are fused and only one lattice is required; however, it is unsuitable for our purposes as it relies on sequential execution. Bailey et al. [2] propose a new algorithm, suitable for parallel operation and tested on a GPU, that uses only one copy of the lattice and achieves performance in between the 'two step' and the 'two lattice' methods. Their algorithm performs the collision step alone for even time steps, odd time steps fuse streaming followed by collision followed by another streaming operation.

We use Nvidia's CUDA language extensions for C to write code for the GPU, implementing domain decomposition in order to use multiple GPUs in a single machine. The machine used for testing has two Nvidia GTX295 graphics cards, each with two GT200 series processors. Each GPU has 896 MB of RAM and 30 multiprocessors, each containing eight stream processors (equivalent to FPUs), resulting in a theoretical performance of 894 Gflops in single precision and 112 GB/sec memory bandwidth.

4.1 Parallel processing

Each element of a calculation has its own thread on the GPU. With an enormous reserve of threads, typically at least tens of thousands, ready to perform work, the GPU's task scheduler keeps the processing units constantly busy while idle threads perform memory transactions. Synchronisation over the entire domain stalls thread scheduling and degrades performance. Nonlocal communication such as dot product or matrix multiplication type operations require data reduction. This increases the complexity of the algorithm and reduces efficiency. The LB method is well suited to both of these concerns. The collision step is fully local and the streaming step requires only nearest neighbour communication. Also, synchronisation is only required once per time step for most algorithms, increasing to twice per time step for the 'two step' method.

The memory controller achieves optimal bandwidth if the data requested by a group of 16 threads (half a warp) is sequential and aligned to 64 byte or 128 byte segments. Our method groups threads into blocks consisting of two parallel stripes of 32 nodes incremented in the primary indexing direction (the x direction), thus allowing all reads to be both aligned and sequential during collision. However, during the streaming step, 10 out of 19 store operations are shifted in the x direction and are no longer aligned. Bailey et al. [2] and Toelke and Krafczyk [14] stored stripes as long as the domain in the x direction into on-die shared memory in order to shift values and restore the alignment for all store operations. This is not feasible for larger domains due to limited shared memory.

The ratio of the number of active threads to the maximum number of threads per multiprocessor (1024 on GT200 hardware), referred to as the occupancy, is also important to thread scheduling. The number of 32 bit registers (16,000) and the amount of shared memory (16 kB) available on each multiprocessor are the most common constraints to the number of active threads. With threads grouped into blocks of 64, our main kernel uses 50 registers per thread and 1332 bytes of shared memory per block using single precision, thus allowing

five active blocks (320 threads). In double precision, it uses 2624 byte of shared memory and only 40 registers allowing six active blocks (384 threads); however, the compiler spills 152 bytes per thread worth of register space into local memory, greatly impacting performance. The compiler spills variables to local memory when arrays are accessed dynamically and when the maximum number of registers is exceeded. There are enough registers available using single precision that we avoid spilling by referring to lattice velocities explicitly.

Domain decomposition is accomplished by slicing in the wall-normal direction, with each slice overlapping its neighbour by one node. Each slice is further subdivided into three sub-slices, which are calculated sequentially. The sub-slices on the edges, which are linked to data on other GPUs, are calculated first and then the boundary communication is performed while the centre sub-slice is computed. Boundary data needs to be passed through the CPU and buffered in the system memory since there is no direct communication between the GPUs.

4.2 Simulation parameters

The flow is defined by the Reynolds number Re_τ based on the friction velocity u_τ. The wall shear τ_w is related to the applied body force f, by a static balance of forces as

$$f\delta = \tau_w = \mu \left.\frac{\partial u}{\partial y}\right|_{y=0}, \tag{6}$$

with u_τ and Re_τ defined in terms of f as

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} = \sqrt{\frac{f\delta}{\rho}} \quad \text{and} \quad \mathrm{Re}_\tau = \frac{u_\tau \delta}{\nu} = \frac{\delta}{\nu}\sqrt{\frac{f\delta}{\rho}}\,. \tag{7}$$

Lattice spacing and time step are both fixed at one in lattice units. The size of the domain δ is determined by the resolution, whereas the viscosity and the body force remain variable.

We specify the Reynolds number, with the added constraint that the Mach number should be below some threshold in order to decrease the compressibility error. The Mach number plays a similar role to the CFL number

$$\mathrm{CFL} = \frac{\Delta t \cdot u_{max}}{\Delta x} . \tag{8}$$

With Δt and Δx both equal to one in lattice units,

$$\mathrm{CFL} = u_{max} , \quad \text{and} \quad \mathrm{Ma} = \frac{u_{max}}{c_s} = \frac{\mathrm{CFL}}{c_s} = \sqrt{3} \times \mathrm{CFL} . \tag{9}$$

We used a Mach number of 0.1, equivalent to a CFL number of 0.06 for all simulations.

5 Results

5.1 Turbulent channel flow

To generate turbulent channel flow, the flow field had to be initialized with a sufficiently large perturbation. A random three dimensional vector field was smoothed to reduce high frequency noise, a potential source of instability. We take the curl of this field to yield a divergence free velocity perturbation that is superimposed on a mean streamwise velocity profile based on the $1/7$ power law. After initialization, the friction velocity first drops and then rises to a peak, corresponding to the onset of turbulence. We let the simulation run for 1000 eddy turnover times after this peak to be certain that the flow is in the fully developed regime,

$$t_{eddy} = \frac{\delta}{u_{max}} . \tag{10}$$

Turbulence statistics are then sampled for a further 1000 eddy turnover times.

The results presented below are obtained on the $l_y = 2$, $l_x = 2\pi$ and $l_z = \pi$ domains summarised in Table 1. Results obtained with the larger domains

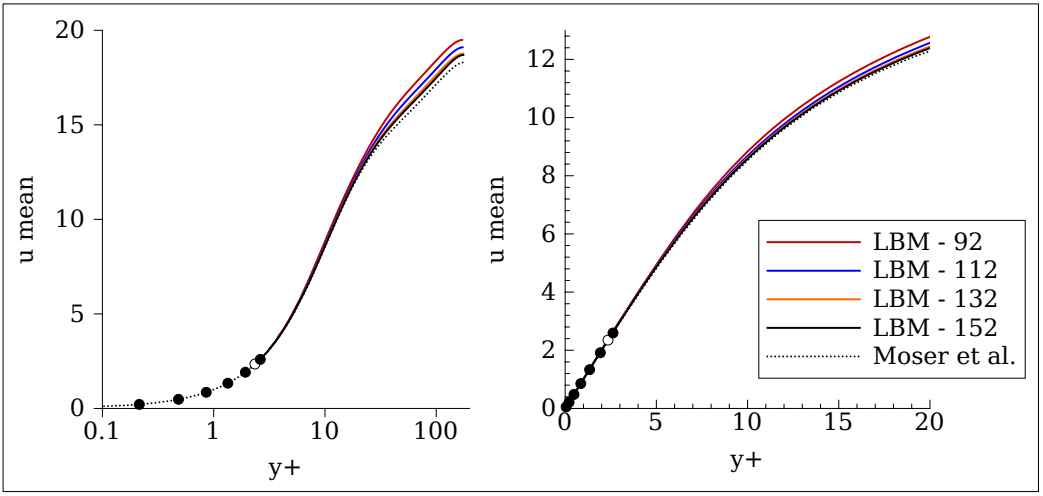

Figure 1: Mean streamwise velocity profiles.

show negligible variation. The largest feasible grid with the large domain is 110 nodes in the wall normal direction, increasing to 150 nodes using the smaller domain. The results were validated against the spectral results of Moser, Kim and Mansour [12].

Figure 1 shows that the mean velocity profile approaches the spectral results as the grid is refined. We see the viscous sublayer where $y+$ is less than five. The log law region has $y+$ greater than 30. The closest node to the wall using the grid with $N_y = 152$ is shown as a white filled dot. The black dots are grid points using the spectral method showing the increased near wall resolution used by Moser and colleagues.

The streamwise fluctuating velocity is the largest contributor to the turbulent kinetic energy (TKE). The height of the peaks in TKE, as shown in Figure 2, is a good indicator of agreement between results, with the results obtained using the LB method approaching the spectral method with increasing resolution. The TKE profile shows better agreement away from peaks. Figure 3 shows the streamwise energy spectrum for $y+ = 19$ with our results agreeing with

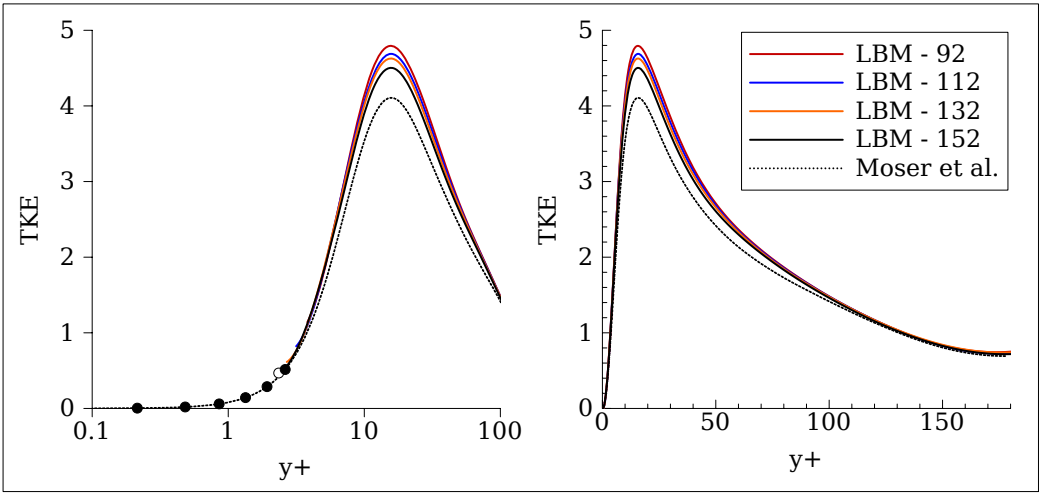

Figure 2: Turbulent kinetic energy profiles.

those of Moser et al. [12] in the inertial range and extending further into the dissipative range.

We avoid numerical instability at $\mathrm{Re}_\tau = 180$ despite reports of stability problems at higher Reynolds numbers using the LB method. The viscosity is kept sufficiently high by the high grid resolution. Experiments with $\mathrm{Re}_\tau = 395$ did prove to be unstable using the largest feasible lattice for our system, 152 nodes in the wall normal direction. If possible, increasing the resolution to 180 nodes in the wall normal direction would increase the viscosity to a value that has been stable in other situations. This minimum resolution is still lower than the 256 nodes used by Moser et al. for the same Reynolds number.

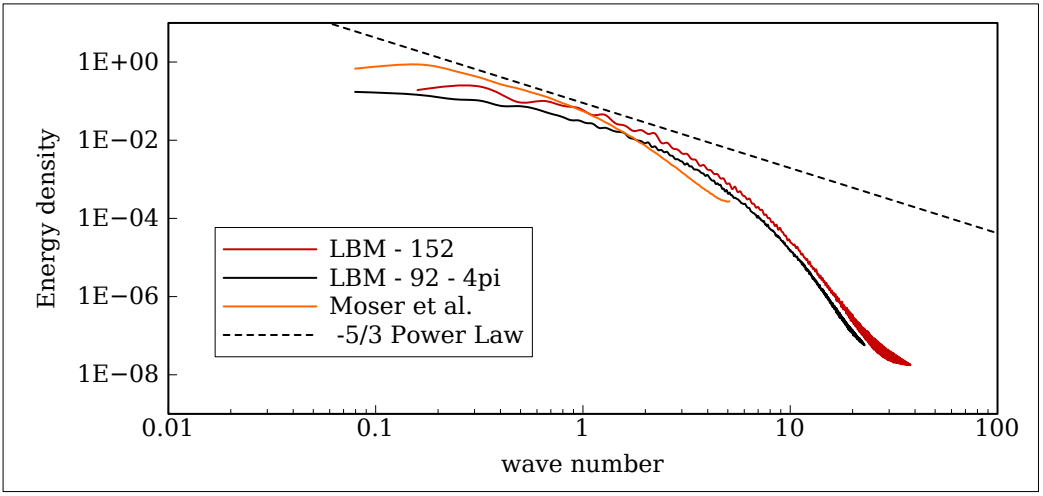

Figure 3: Streamwise energy spectrum for $y+ = 19$.

5.2 Computational performance

Table 2 shows performance results, measured in Mlups (million lattice updates per second) for various configurations tested. The first three entries in the table are performance benchmarks obtained by iterating the LB kernel alone, with the fourth entry 'two step full' demonstrating the 30% reduction in performance after adding boundary conditions and sampling turbulence statistics.

Performance per processor using domain decomposition and four GPUs is shown in the final entry 'two step multi', with parallel efficiencies of 65% and 88% using single and double precision respectively. The main issue degrading performance when using domain decomposition is the bandwidth available for communication of boundary information between GPUs.

Bailey et al. [2] and Toelke and Krafczyk [14] used a Nvidia 8800 Ultra which has a memory bandwidth of 104 GB/sec and computational throughput of 576 Gflops compared to the 112 GB/sec and 894 Gflops of the GTX295 used in

Table 2: Computational performance.

	Single precision			Double precision		
Algorithm	Mlups	Gflops	GB/sec	Mlups	Gflops	GB/sec
two lattice ⋆	358	91.3	54.4	91	23.2	27.7
two lattice	186	47.4	28.3	-	-	-
two step simple	101	25.8	30.7	55	14.0	33.4
two step full	71	18.1	21.6	42	10.7	25.5
two step multi	46	11.7	14.0	37	9.4	22.5

our work. Bailey et al. [2] achieved an impressive 300 Mlups using a 'two lattice' algorithm and 260 Mlups using their improved single lattice method. Toelke and Krafczyk achieve up to 582 Mlups using the less computationally intense D3Q13 lattice and MRT collision operator. They both use a technique to make all memory stores aligned. As detailed in Section 4.1, this optimization is not feasible for larger domains. The 'two lattice ⋆' method shows the performance of our kernel if all stores are artificially forced to be aligned; when this is done our method has a speed comparable to that of Bailey et al. [2].

Table 3 gives the performance of the present algorithm, implemented using standard C and parallelized using OpenMP, for comparison with GPU performance. We tested two processors: an Intel i7 920 overclocked to 3.2 GHz; and an AMD X6 1090T at the standard clock speed of 3.2 GHz. The memory of both systems is set to 2 GHz. The GPU achieves a minimum speed up factor of around 20.

We also tested the less computationally intense D3Q15 lattice and encountered the chequerboard instability as shown in Figure 4 and documented by Kandhai et al. [8]. The D3Q19 lattice does not show these effects and so is used for all our simulations.

Table 3: CPU performance

		Single precision			Double precision		
CPU	Threads	Mlups	Gflops	GB/sec	Mlups	Gflops	GB/sec
AMD X6 1090T	1	0.66	0.17	0.20	0.52	0.13	0.32
AMD X6 1090T	4	2.08	0.53	0.63	1.75	0.45	1.06
AMD X6 1090T	6	2.44	0.62	0.74	2.14	0.55	1.30
Intel i7 920	1	0.89	0.23	0.27	0.84	0.21	0.51
Intel i7 920	4	3.22	0.82	0.98	2.99	0.76	1.82

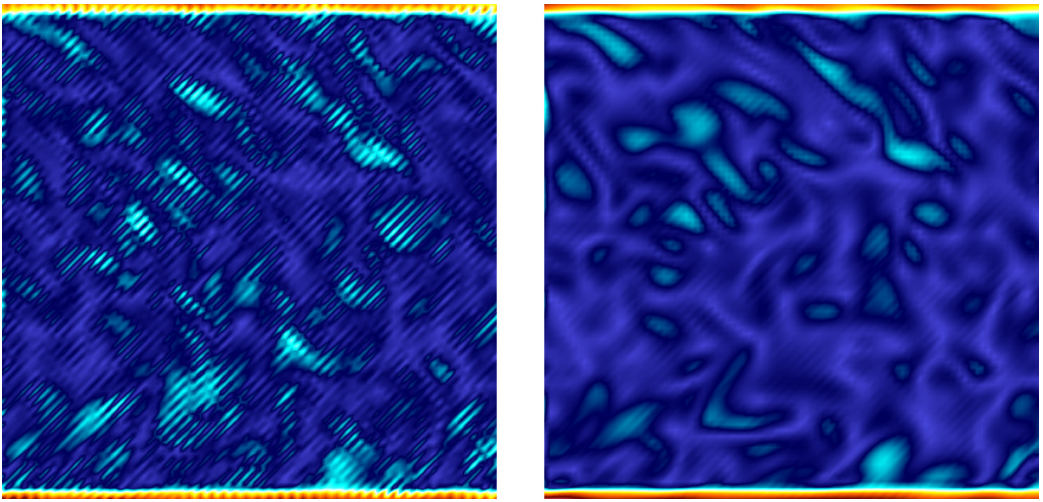

Figure 4: Vorticity magnitude at the same simulation time using the D3Q15 lattice (left) and the D3Q19 lattice (right).

6 Conclusion

The LB method has been successfully used to simulate turbulent channel flow. The main errors stem from the inability to capture near wall gradients due to the uniform lattice. The method may be more suitable for studying turbulent flows that are not bounded by walls. One option to make the method practical for simulation of wall bounded flows would be to use embedded high resolution grids close to the walls. This is possible with the LB method.

We achieved an efficient parallel implementation due to the explicit and local nature of the LB method. Using GPU computing, we achieved a much higher performance than possible using standard CPUs. The performance levels reached were lower than some published results due to a different choice of algorithm and our code having additional features to study fully turbulent flow. Our code also scales well to multiple GPUs using domain decomposition.

Acknowledgements: This work was supported by the Australian Research Council and by an Australian Postgraduate Award.

References

[1] Cyrus K. Aidun and Jonathan R. Clausen. Lattice Boltzmann method for complex flows. *Annu. Rev. Fluid Mech.*, 42:439–472, 2010. doi:10.1146/annurev-fluid-121108-145519 C916

[2] P. Bailey, J. Myre, S. D. C. Walsh, D. J. Lilja, and M. O. Saar. Accelerating lattice Boltzmann fluid flow simulations using graphics processors. In *International Conference on Parallel Processing, ICPC '09*, pages 550–557. Sept 2009. doi:10.1109/ICPP.2009.38 C916, C919, C920, C925, C926

[3] R. Benzi, S. Succi, and M. Vergassola. The lattice Boltzmann equation—theory and applications. *Phys. Rep.*, 222(3):145–197, Dec 1992. doi:10.1016/0370-1573(92)90090-M C917

[4] Massimo Bernaschi, Massimiliano Fatica, Simone Melchionna, Sauro Succi, and Efthimios Kaxiras. A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. *Concurrency Computat. Pract. Exper.*, 22(1):1–14, Jan 2010. doi:10.1002/cpe.1466 C916

[5] Dustin Bespalko, Andrew Pollard, and Mesbah Uddin. Direct numerical simulation of fully-developed turbulent channel flow using the lattice Boltzmann method and analysis of OpenMP scalability. In *High Performance Computing Systems and Applications*, volume 5976 of *Lect. Notes Comput. Sci.*, pages 1–19. Springer Berlin, 2010. doi:10.1007/978-3-642-12659-8_1 C916

[6] Z. L. Guo, C. G. Zheng, and B. C. Shi. An extrapolation method for boundary conditions in lattice Boltzmann method. *Phys. Fluids*, 14(6):2007–2010, Jun 2002. doi:10.1063/1.1471914 C918

[7] M. Junk, A. Klar, and L. S. Luo. Asymptotic analysis of the lattice Boltzmann equation. *J. Comput. Phys.*, 210(2):676–704, Dec 10 2005. doi:10.1016/j.jcp.2005.05.003 C917

[8] D. Kandhai, A. Koponen, A. Hoekstra, M. Kataja, J. Timonen, and P. M. A. Sloot. Implementation aspects of 3D lattice-BGK: Boundaries, accuracy, and a new fast relaxation method. *J. Comput. Phys.*, 150(2):482–501, Apr 10 1999. doi:10.1006/jcph.1999.6191 C926

[9] Waleed Abdel Kareem, Seiichiro Izawa, Ao-Kui Xiong, and Yu Fukunishi. Lattice Boltzmann simulations of homogeneous isotropic turbulence. *Comput. Math. Appl.*, 58(5):1055–1061, Sep 2009. doi:10.1016/j.camwa.2009.02.002 C916

[10] Keijo Mattila, Jari Hyvaeluoma, Jussi Timonen, and Tuomo Rossi. Comparison of implementations of the lattice-Boltzmann method. *Comput. Math. Appl.*, 55(7):1514–1524, Apr 2008. doi:10.1016/j.camwa.2007.08.001 C919

[11] Keijo Mattila, Jari Hyvaluoma, Tuomo Rossi, Mats Aspnas, and Jan Westerholm. An efficient swap algorithm for the lattice Boltzmann method. *Comput. Phys. Commun.*, 176(3):200–210, Feb 1 2007. doi:10.1016/j.cpc.2006.09.005 C919

[12] R. D. Moser, J. Kim, and N. N. Mansour. Direct numerical simulation of turbulent channel flow up to Re_τ=590. *Phys. Fluids*, 11(4):943–945, Apr 1999. doi:10.1063/1.869966 C918, C923, C924

[13] Yan Peng, Wei Liao, Li-Shi Luo, and Lian-Ping Wang. Comparison of the lattice Boltzmann and pseudo-spectral methods for decaying turbulence: Low-order statistics. *Comput. Fluids*, 39(4):568–591, Apr 2010. doi:10.1016/j.compfluid.2009.10.002 C916

[14] J. Toelke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *Int. J. Comput. Fluid Dyn.*, 22(7):443–456, 2008. doi:10.1080/10618560802238275 4th International Conference for Mesoscopic Methods in Engineering and Science, Munich, Germany, Jul 16–20, 2007. C916, C920, C925

[15] D. A. Wolf-Gladrow. *Lattice-gas cellular automata and lattice Boltzmann models—Introduction*, volume 1725 of *Lect. Notes Math.* Springer Berlin, 2000. C917

Author addresses

1. **V. Zecevic**, School of Aerospace, Mechanical & Mechatronic Engineering, University of Sydney, NSW, AUSTRALIA.
mailto:vanja.zecevic@sydney.edu.au

2. **M. P. Kirkpatrick**, School of Aerospace, Mechanical & Mechatronic Engineering, University of Sydney, NSW, Australia.
mailto:michael.kirkpatrick@sydney.edu.au

3. **S. W. Armfield**, School of Aerospace, Mechanical & Mechatronic Engineering, University of Sydney, NSW, Australia.
mailto:steven.armfield@sydney.edu.au