International Journal of Heat and Mass Transfer 116 (2018) 715–724

Contents lists available at ScienceDirect

International Journal of Heat and Mass Transfer

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

Numerical predictions of laminar and turbulent forced convection: Lattice Boltzmann simulations using parallel libraries

Mehaboob Basha [a], Nor Azwadi Che Sidik [a,b,*]

[a] Department of Thermo-fluid, Faculty of Mechanical Engineering, Universiti Teknologi Malaysia, Johor, Malaysia
[b] Malaysia – Japan International Institute of Technology (MJIIT), University Teknologi Malaysia Kuala Lumpur, Jalan Sultan Yahya Petra, 54100 Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 6 March 2017
Received in revised form 15 September 2017
Accepted 18 September 2017
Available online 22 September 2017

Keywords:
Parallel lattice Boltzmann method
Domain-decomposition
Matlabpool
MPI
OpenMP
OpenMPI

ABSTRACT

This paper presents the performance comparison of various parallel lattice Boltzmann codes for simulation of incompressible laminar convection in 2D and 3D channels. Five different parallel libraries namely; matlabpool, pMatlab, GPU-Matlab, OpenMP and OpenMP+OpenMPI were used to parallelize the serial lattice Boltzmann method code. Domain decomposition method was adopted for parallelism for 2D and 3D uniform lattice grids. Bhatnagar-Gross-Krook approximation with lattice types D2Q9, D2Q19 and D2Q5, D2Q6 were considered to solve 2D and 3D fluid flow and heat transfer respectively. Parallel computations were conducted on a workstation and an IBM HPC cluster with 32 nodes. Laminar forced convection in a 2D and turbulent forced convection in a 3D channels was considered as a test case. The performance of parallel LBM codes was compared with serial LBM code. Results show that for a given problem, parallel simulations using matlabpool and pMatlab library perform almost equal. Parallel simulations using C language with OpenMP libraries were 10 times faster than simulations involving Matlab parallel libraries. Parallel simulations with OpenMP+OpenMPI were 0.35 times faster than the reported parallel lattice Boltzmann method code in the literature.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

From past two decades, lattice Boltzmann method in conjunction with single relaxation collision operator [1–5] is widely used to simulate dynamics of mesoscopic fluid flow and heat transfer system through fictitious particles collision and redistribution on a lattice grid with pre-defined lattice velocities. Under a low Mach number assumption, Chapman-Enskog analysis [6] of LB equation associates moments of equilibrium particles to physical (macroscopic) fluid flow variables, such as density, velocity, temperature, etc., in Navier-Stokes equations. Easy handling of complex boundary, simplicity, accuracy [7–9], etc., has led to application of LBM for solving wide variety of fluid flow and heat transfer problems [10–14].

However, the main disadvantage of LBM is that it is computationally intensive. For instance, LBM simulation of two and three-dimensional fluid flow problems requires 9 and 19 lattice velocities (D2Q9 and D3Q19) at every grid point, respectively. Moreover, for stable and accurate LBM simulation, lattice nodes should be scaled with Reynolds number and domain size, such that Mach number (in lattice units) is less than 0.3. Hence, for simulation of high Reynolds number fluid flow or fluid flow in large domain or both, results in large lattice grid size (large data arrays). A serial LBM code could take months or weeks, if not days to get converged solution for large data arrays. Since, the moments of particles distributions functions are local in nature for calculation of fluid flow variables, such as density, velocity, temperature, etc., parallelization of LBM is relatively easy [7].

To improve the performance of the LBM code and to reduce the simulation time, several techniques are proposed and implemented in the literature. One of the techniques is data parallelism [15], where large data arrays of the problem are decomposed into several small subsets that are computed in parallel on multi-core processor of a computer. Another technique is a grid refinement [16], where fine grid is adopted in the critical regions, such as near wall, high gradient regions, etc. and coarse grid is adopted in non-critical regions of the flow domain. Use of local grid refinement or non-uniform grid not only reduces memory size but also reduces computational time. However, numerical error is inevitable during interpolation of particle distribution functions in grid refinement techniques [16].

Following is the literature review on parallel simulations using LBM. Satofuka and Nishioka [15] used parallel technique to solve 3D incompressible turbulent flow using LBM. Derksen and Van

* Corresponding author at: Department of Thermo-fluid, Faculty of Mechanical Engineering, Universiti Teknologi Malaysia, Johor, Malaysia.
E-mail address: azwadi@mail.fkm.utm.my (N.A.C. Sidik).

https://doi.org/10.1016/j.ijheatmasstransfer.2017.09.072
0017-9310/© 2017 Elsevier Ltd. All rights reserved.

Nomenclature

BC	boundary condition
BGK	Batnaghar, Gross, Krook
c	lattice velocities
CPU	central processing unit
Cs	speed of sound
Cp	specific heat
Dh	hydraulic diameter
2D	two dimensional
3D	three dimensional
f_i, g_i	particle distribution function
GPU	graphics processing unit
HPC	high performance computing
L	length of the channel
LBM	lattice Boltzmann method
Ma	mach number
MPI	message passing interface
NS	Naiver-Stokes
N_p	number of process
p	pressure
P	process
x, y, z	co-ordinates
u, v, w	velocities in x, y, z direction, respectively
w_i	lattice weights

Subscript

d	dimensionless
i	ith direction
id	ID of processor
eq	equilibrium
neq	non-equilibrium
o	reference condition
p	process
tb	turbulence

List of symbols

δ_x	lattice size, m
δ_t	lattice time, s
ρ_o	mean/reference density, kg/m^3
ω	inverse of relaxation time, 1/s
μ	dynamic viscosity, kg m/s^2
μ_t	turbulent dynamic viscosity, kg m/s^2
υ	kinematic viscosity, m/s^2
υ_t	turbulent kinematic viscosity, m/s^2
τ	relaxation time, s
τ_*	total relaxation time, s
τ_t	turbulent relaxation time, s
S	strain rate tensor
Π	stress tensor
Π^{neq}	non-equilibrium stress tensor

den Akker [17] performed SGS Large eddy simulations of turbulent fluid flow in a baffled stirred tank driven by a Rushton turbine by applying LBM. Equivalent body force was applied for representing the action of the impeller on the fluid. The parallel simulations were conducted on a shared-memory architecture computer. Cherba et al. [18] presented performance analysis of a parallel 2D LBM on various configurations of cluster computers. Results indicated that increase in data precision does not affect execution time significantly on Pentium class processors. Study also showed that improved communication and calculation strategies can yield better speedup and scalability. A massively parallel code for particle suspension problems using the LBM was presented by Stratfrord et al. [19]. This paper compares performance of the code in terms of the computational overhead required for the particle laden flow problem with the fluid-only problem, and for the scaling of the code to large processor numbers. Various parallel techniques to increase the single-CPU performance, and the impact on the parallelization techniques on performance were presented by Carolin et al. [20]. The parallel techniques were applied to solve fluid flow involving free surfaces and also the paper discusses about the required extensions to handle complex flow scenario. Data blocking parallel implementation of 2D and 3D Lattice Boltzmann Method was presented by Claudio et al. [21]. Their results showed that blocked parallel implementation can enhance performance up to 31% than non-blocked versions of the LBM code. Dustin et al. [22] performed DNS simulation of turbulent 3D periodic channel using LBM with multiple relaxation time in collision process. The parallel computations were conducted on 256 processors shared memory machine using OpenMP. Computational time per iteration was found to be less than 0.5 s for a grid size of ($91 \times 181 \times 1080 \times 19$ lattice velocities = 337984920 data-size). Florian et al. [23] presented algorithms for non-uniform grid, large-scale, massively parallel LB-based simulations on distributed data structures for waLBerla software. Their algorithm on an IBM Blue Gene/Q system, gave perfect scalability with absolute performance of close to a trillion node updates per second, while on an Intel-based system, an absolute performance of 8.5 million node updates per second was obtained.

Computer languages such as C, C++ and FORTRAN are used worldwide for coding serial and parallel LBM codes [17–22]. Recently, GPU computing with CUDA has received lot of attention from researchers for parallel LBM simulations [24]. However, coding and debugging in the above mentioned languages is quite tedious and time consuming task, especially, when dealing with CUDA codes. From couples of years MATLAB is being used for technical computing due to availability of several ready-to-use built-in libraries [25]. It can also be used for rapid prototyping of pilot codes and then translate to C or FORTRAN code. Moreover, parallel libraries such as Parallel toolbox in MATLAB and pMatlab by Lincoln laboratories, MIT [26], can be used to build Parallel LBM code easily.

Therefore, the objectives of this study are to build parallel LBM codes using Matlab parallel library and subsequently rewrite the parallel Matlab code in C language with OpenMP and OpenMPI libraries, and also to compare the performance of the parallel codes with performance of serial code. As a test case, incompressible convection in 2D and 3D channels is considered, in conjunction with stable fluid flow [27] and thermal boundary conditions [10].

2. Methodology

2.2. Numerical method

Incompressible LBGK model proposed by He and Lou [7] is adopted here. In LBM, space is discretized into uniform lattice size of δ_x and velocity is discretized into finite number of velocities $\vec{c}_i$ to form particle distribution functions $f_i(\vec{r}, t)$. The LBGK evolution equation is as follows.

$$f_i(\vec{r} + \delta_x \vec{c}_i, t + \delta_t) - f_i(\vec{r}, t) = -\Omega_i, \quad \Omega_i = -\omega(f_i - f_i^{eq}) + FT_i \tag{1}$$

In Eq. (1) Ω_i is the BGK collision operator which defines particle interaction on lattice sites. FT_i is represents body force. Flow dynamics evolve through series of collision and streaming of particle distribution functions. During each time step before collision, particle distribution functions are regularized following the method in Ref. [27]. Macroscopic variables of the flow are recovered by the moments of particle distribution functions as $\rho = \sum_{i=1} f_i^{eq}$, $\rho_o\, u_\alpha = \sum_i c_{i\alpha} f_i^{eq} + \frac{1}{2}\delta_t \boldsymbol{F_i}$ and $\Pi_{\alpha\beta} = \sum_i e_{i\alpha} e_{i\beta} f_i$ and equilibrium distribution function is obtained by expansion of Maxwell-Boltzmann equation to second order and it reads as $f^{eq} = \rho w_i \left(1 + \frac{1}{c_s^2} c_i \cdot \boldsymbol{u} + \frac{1}{2c_s^2} Q_i : \boldsymbol{uu}\right)$. Non-equilibrium stress tensors are needed during evolution and are calculated as $\Pi^{neq}_{\alpha\beta} = \Pi_{\alpha\beta} - \Pi^{eq}_{\alpha\beta}$, D2Q9 lattice vectors and weights w_i can be found in [11]. Relation between moments of particle distribution functions and macroscopic fluid flow variables can be established through multi-scale Chapman-Enskog analysis of Eq. (1), in which zeroth order term of particle distribution function is equal to equilibrium distribution function ($f^{(0)} = f^{eq}$) and first order term of particle distribution function through regularization $\left[f^{neq} \approx f^1 = \frac{w_i}{2c_s^2} Q_{i\alpha\beta} \Pi^{neq}_{\alpha\beta} + \frac{w_i}{2c_s^2} Q_{i\alpha\beta}(\boldsymbol{Fu} - \boldsymbol{uF})\right]$ is related to momentum flux tensor at low Mach number [11]. Velocity, Mach number, pressure and Kinematic viscosity of the fluid are related to lattice variable as in Eq. (2).

$$u = \frac{\delta_t}{\delta_x},\ M = \frac{u}{c_s^2},\ P = \rho\ c_s^2 \text{ and } \nu = c_s^2\left(\frac{1}{\omega} - \frac{1}{2}\right) \tag{2}$$

For incompressible fluid, viscous heat dissipation and compression work by pressure are negligible and hence temperature can be treated as a passive scalar that is advected by the flow field. LB equation for scalar transport is given by Eq. (3).

$$g_i(\vec{r} + \delta t \vec{c}_i, t + \delta t) - g_i(\vec{r}, t) = -\omega_t (g_i - g_i^{eq}),\ \omega_t = \frac{1}{\tau_t} \tag{3}$$

where i is lattice direction, and ω_t is inverse relaxation time of energy density distribution function. The Champan-Enskog expansion establishes link between thermal diffusivity and inverse relaxation time of energy density distribution function and is as follows.

$$\alpha_d = c_s^2\left(\frac{1}{\omega_t} - \frac{1}{2}\right) \tag{4}$$

Equilibrium distribution function is obtained by expansion of Maxwell-Boltzmann equation to first order accuracy and it reads as in Eq. (5).

$$g^{eq} = \rho w_i \left(1 + \frac{1}{c_s^2} c_i \cdot \boldsymbol{u}\right) \tag{5}$$

Temperature is obtained by moments of energy distribution functions.

$$T = \sum_i f_i^{eq} \tag{6}$$

To solve the temperature field using LBGK equation, lattice type D2Q5 and D2Q6 models are used for 2D and 3D computational domain, respectively.

2.3. SGS LES turbulence model

Large eddy simulation computes fluid flow motion by resolving large eddies that can affect the fluid flow appreciably. Conventional numerical methods uses filter form of governing NS equations for LES of turbulent fluid flows. Similarly, LES of turbulent fluid flow can be simulated applying filtered form of LBM. The filter form of the LBE for LES is given below [28].

$$\bar{f}_i(x + c_i\delta_t, t + \delta_t) = \bar{f}_i(x, t) - \frac{1}{\tau_*}(\bar{f}_i - \bar{f}_i^{eq}) + FT_i \tag{7}$$

where $\bar{f}_i$ and $\bar{f}_i^{eq}$ are instantaneous particle distribution functions of resolved fluid flow with large scale eddies. The effect of unresolved small scales fluid flow eddies is accounted through an eddy viscosity model via relaxation time, τ_t. Thus the total relaxation time for LES simulation should $\tau_* = \tau_o + \tau_t$, τ_o and τ_t are the relaxation times corresponding to the molecular υ_o and turbulent viscosity υ_t, respectively. Accordingly υ_* is given by 3.8.

$$\upsilon_* = \upsilon_o + \upsilon_t = \frac{1}{3}\left(\tau_* - \frac{1}{2}\right)c^2\delta_t = \frac{1}{3}\left(\tau_o + \tau_t - \frac{1}{2}\right)c^2\delta_t,\ \upsilon_t\ : = \frac{1}{3}\tau_t c^2 \delta_t \tag{8}$$

where υ_t depends on the sub grid model used in the simulation. We use the Smagorinsky model for subgrid closure. In the Smagorinsky model, the eddy viscosity is calculated from the filtered strain rate tensor $S_{\alpha\beta} = \frac{1}{2}(\partial_\alpha u_\beta + \partial_\beta u_\alpha)$ with filter size Δ_x equal to δ_x

$$\upsilon_t = (C_{sm}\Delta_x)^2 \bar{S} \tag{9}$$

$$\bar{S} = \frac{\bar{\Pi}}{2\rho_o C_s^2 \tau_*} \tag{10}$$

$$\bar{\Pi} = \sqrt{2\sum \Pi_{\alpha\beta}\Pi_{\alpha\beta}} \tag{11}$$

where $\bar{S}$ and $\bar{\Pi}$ are the characteristic filtered rate of strain and filtered mean momentum flux, respectively, and C_{sm} is the Smagorinsky constant. Since $\tau_* = \tau_o + \tau_t$ and $\tau_o = 3\upsilon_o + \frac{1}{2}$ and $\tau_t = 3\upsilon_t$ in lattice units. Eq. (9) leads to a quadratic equation for τ_t with C_{sm} and Δ_x as in Eq. (12).

$$\tau_t = \frac{1}{2}\left(\sqrt{\tau_o^2 + 2\sqrt{2}(C_{sm}\delta_x)^2(\rho_o c_s^4 \delta_t)^{-1}\bar{\Pi}} - \tau_o\right) \tag{12}$$

2.4. Parallel methods

Fig. 1 shows the algorithm for serial code. The algorithm starts with allocation of data arrays and then cycles of processes, such as calculation of flow variables, setting boundary conditions, collision and streaming. Modern computers have share memory architecture. Data created/stored in shared memory can be accessed by

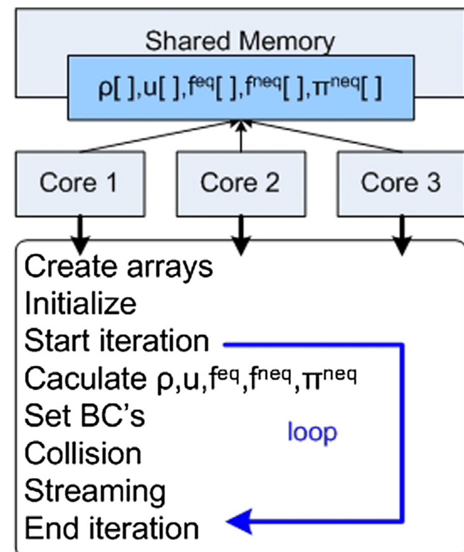

Fig. 1. Serial LBM simulation on a single computer node.

all cores of the computer. In serial code, after submission of the job, iteration can occur on any core of the computer and user does not have explicit control over it.

Typically, for stable and accurate LB-BGK simulation, lattice nodes should scale with Reynolds number and hence large number of lattice nodes are required for simulation of high Reynolds number flows. Mach number is function of lattice size δx and lattice time δt which should be less then 0.3. Actually, LBM simulates pseudo incompressible or weakly compressible fluid flow. To reduce the inherent compressibility effect, lattice size δx should be proportional to square the order of lattice time δt. A multi-core workstation or single node of HPC cluster can handle a moderate number of lattice nodes (data array size) and also take more time for simulation. To reduce the time of simulation, it is necessary to use parallel techniques to gain high performance from available multi-core workstation or multi-node HPC cluster. In LBM, evolution of field takes place in two consequent steps namely collision and streaming. Due to this, task parallelization is not possible. However, most of the LBM simulation are data intensive (large data array size) and can be parallelized. It is also possible to combine collision and streaming steps to reduce processing time there by increasing performance [29]. However, the present study uses data or domain decomposition method to achieve parallelism [24].

Schematic of the domain decomposition method is presented in Fig.2a. For simplicity and demonstration purpose, flow in a 2D channel is considered. The 2D channel is divided into three blocks and each block is discretized into 5 × 5 lattices nodes. Fig.2b shows multi-core parallel simulation algorithm on a single computer node using Matlab's Matlabpool library. The parallel algorithm distributes a block to each core in the multi-core workstation, and all cores will work synchronously on respective blocks. This type of parallelism is called as single-program multiple data (SPMD). As mentioned earlier, in LBM simulation the flow field evolves through collision and streaming of particle distribution function on the lattice nodes. During streaming process each particle distribution function migrates from its node to nearest corresponding node in the same direction. After every time step, some of the particle distribution function (red[1] colored distribution functions) at the boundaries are missing and these particle distribution functions should stream-in from corresponding neighboring blocks. This exchange of information is handled by message passing interface (MPI calls). During message passing process, each message carries required particle distribution function, a address as to where the message has to be sent, and a tag which acts as identifier. Once the simulation is converged all data is gathered to regular full sized data array. Due to domain decomposition and message passing calls, streaming process requires additional mapping of particle distribution at the internal boundaries (red colored distribution function in Fig.2a).

In Matlab, matlabpool is a parallel library that can create a pool of multiple processes on a local computer node. In the algorithm, syntax SPMD, create and runs a copy of the code on all processes. Computational data (PDF's f_i, f_i^{neq}, etc.,) can be distributed either by creating a shared public full sized arrays and accessing a portion of it from each core/process or by creating a own copy of private data (PDF's f_i, f_i^{neq}, etc.,) array on every core of the computer node. Obviously, for a given simulation, full size shared arrays occupy large memory space and private data arrays occupy small memory space. Hence, computation using private data arrays are faster

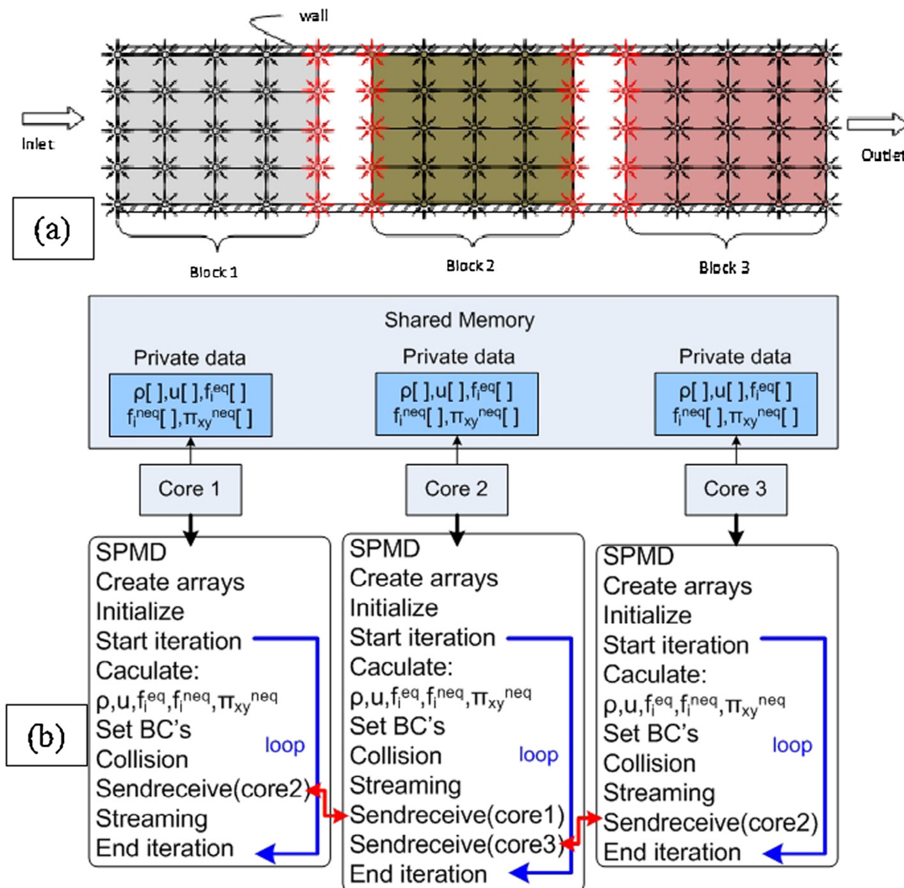

Fig. 2. Parallel LBM simulation on a single computer node using Matlab. (a) schematic of 2D lattice grid decomposition, (b) Algorithm.

[11] due to higher bandwidth. Present study uses private data arrays for all parallel simulations including Matlabpool, pMatlab, OpenMP, OpenMPI/Intel-MPI libraries.

A generalized message passing code segment for domain decomposition method is shown Fig. 3. It can be seen in Fig. 3 that first and last block has one receive and one send message passing call. This is because each block has one neighboring boundary that sends are receives particle distribution function mutually. Rest of the intermediate blocks have two send and receive message passing calls each, which is due to the fact that each block has two boundaries which require particle distribution functions from neighboring blocks.

Fig. 4 shows multi-core parallel simulation algorithm on a single computer node using OpenMP library in C language. OpenMP is a parallel library (compiler directives) when used, can create a pool of multiple processes on a local computer node with shared memory architecture. In the algorithm, syntax *# pragma omp parallel*, creates and runs a copy of the code on all processes. Computational data can be distributed either by creating a shared public full sized arrays or by creating a own copy of private data array on every core of the computer node. Private data arrays are used for simulations. Each core/process will work on its own data arrays synchronously. Once the iterations are complete, data from private arrays from all cores/process is gathered to a full size arrays on master thread. Unlike in SPMD matlab simulations, after every time step or iteration, right after collision, halo data is stored to a shared buffer array and copied to the adjacent core/process. Number of shared buffer arrays and its size will depend on lattice type and number of processes used in the simulation. For D2Q9 lattice, three particle distribution functions migrate from either sides of the block. Hence three buffer arrays are required at each boundary of the block, i.e., three *buffer-in [No. of cores]* and *buffer-out[No. of cores]* arrays each are required for 2D fluid flow simulations. Like wise, for D3Q19, i.e., six *buffer-in [No. of cores]* and *buffer-out[No. of cores]* arrays each are required for 3D fluid flow simulations. Fig. 4 shows two buffer arrays namely, *buffer-in* and *buffer-out* in the shared memory region with each process/core writing to it. The *buffer-in* array is for storing particle distribution functions from inlet boundary points of the data block, while *buffer-out* array is for storing particle distribution functions from outlet boundary points of the data block. After streaming, particle distribution functions from the buffers are copied to respective adjacent block or core. At the end of simulation, required data such as velocities, temperature, etc., are written to a file.

Fig. 5 shows multi-core multi-node parallel simulation algorithm for a HPC cluster using OpenMP and Intel-MPI/OpenMPI library in C language. Intel-MPI/OpenMPI is a parallel library when used with C language can create a pool of multiple processes on a local computer node or HPC cluster with distributed memory architecture. In the algorithm, syntax *# pragma omp parallel*, creates and runs a copy of the code for a given number of processes on local node with shared memory architecture, while syntax *MPI-INIT()* creates and runs a copy of the code for a given number of distributed nodes with distributed memory architecture. OpenMP library will work on private data arrays of local node during simulations, while OpenMPI/Intel-MPI library will just distribute the work and communicate messages between nodes during simulations. Again here, each core/process will work on its own data arrays synchronously. Once the iterations are complete, data from private arrays from local cores/process is gathered to a full size distributed arrays on OpenMP master thread of respective local nodes and then these distributed arrays are

```
%  first block
if (Pid = 0 & Np>1) % Pid is processor Id corresponds to block number.
    send_order=Pid+1; % sending message to which block.
    rec_order=Pid+1;        % receiving message from which block.
    varout=fOutloc(:,out,:); % distribution functions to send.
    SendMsg(send_order,tag,varout); % message passing
    varin=RecvMsg(rec_order,tag); % message passing
    fEqloc(:,out,:)=varin; % distribution functions to receive
% intermediate blocks
  elseif (Pid > 0 && Pid <= Np-2)
    send_order=[Pid-1,Pid+1];       rec_order=[Pid-1,Pid+1];
    varout=fOutloc(:,1,:);
    varout1=fOutloc(:,out,:);
    SendMsg(send_order(1),tag,varout);
    SendMsg(send_order(2),tag,varout1);
    varin=RecvMsg(rec_order(1),tag);
    varin1=RecvMsg(rec_order(2),tag);
    fEqloc(:,1,:)=varin;     fEqloc(:,out,:)=varin1;
% last block
  elseif (Pid == Np-1 && Np >1)
    send_order=Pid-1;       rec_order=Pid-1;
    varout=fOutloc(:,1,:);
    SendMsg(send_order,tag,varout);
    varin=RecvMsg(rec_order,tag);
    fEqloc(:,1,:)=varin;
  end
tag = mod(tag+1,32)+1; % tag increment
```

Fig. 3. Code segment for message passing between multiple block/multiple core.

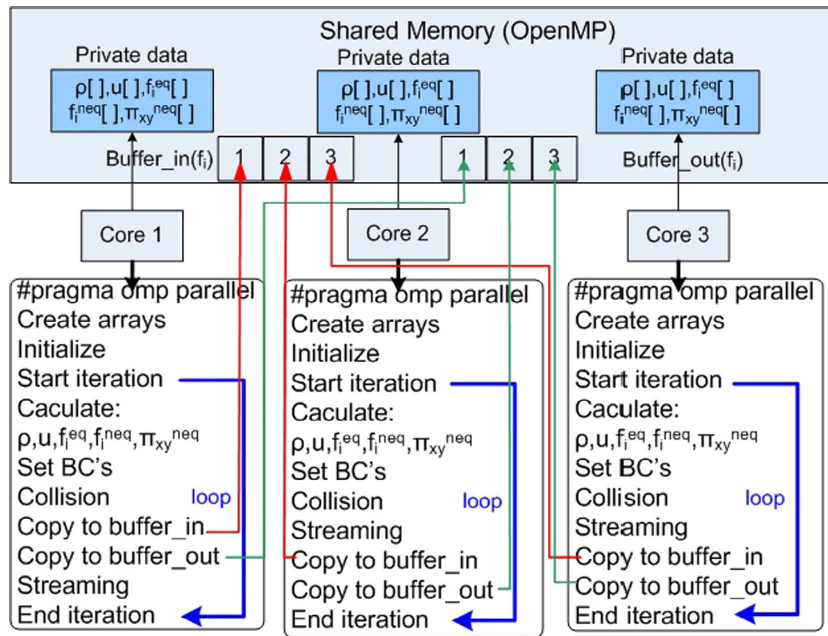

Fig. 4. Multi-core parallel LBM simulation on a single computer node using OpenMP.

gathered to a global array on OpenMPI Master thread by Message passing calls. Since it involves combination of shared and distributed memory computing, copying of particle distribution function to a buffer arrays and message passing call are required. After every time step or iteration, within the local node, right after collision, halo data is stored to a shared buffer array and copied to the adjacent core/process. Whereas adjoining blocks on non-local nodes exchange halo data information by MPI calls. Since each node has one or two OpenMPI thread, all OpenMPI threads will send and receive messages during simulation. At the end of simulation, required data such as velocities, temperature, etc., are written to a file from a master node.

3. Results and discussion

Previous section presented parallel technique for improving the performance of Matlab and C LBM codes. This section presents evaluation of above mentioned techniques. Fig. 6 show the schematics of 2D and 3D computational domain used for parallel simulations. At the outlet zero gauge pressure is assigned by setting mean density to be equal to 1. No-slip velocity condition is assigned at walls. These macroscopic fluid flow boundary conditions are casted into regularized particle distribution functions following the procedure of Latt et al. [27] and thermal boundary conditions are casted into particle distribution functions following Nor Azwadi and Tanahashi [11] method.

Before implementing the parallel technique, serial LBM code is vectorized to further reduce the simulation time. Code vectorization is achieved by simply assigning a individual array to each of the lattice vectors. In addition, several in-line calculation of the code is restructured as array functions. Serial LBM code and vectorized serial LBM code are used to simulate two identical 2D channel flow with grid size of 100×600. Elapsed time for serial code with and without vectorization for a given fluid flow problem for 1000 iterations is found to be 220 and 105 s, respectively. This implies that vector serial code is twice faster than non-vector serial code.

Initially, Matlab is used to develop parallel LBM code. Vector serial code is parallelized using three libraries namely, parallel computation toolbox, Matlabpool in MATLAB, parallel library, pMatlab developed by MIT and GPU library in MATLAB. Parallel algorithm for the Matlabpool and pMatlab is almost identical, while parallel algorithm in MATLAB GPU is in-built and user does not have access to the algorithm. The parallel RLBM algorithms are used to simulate laminar fluid flow in a 2D channel. Reynolds number based on hydraulic diameter was kept constant at 100. Aspect ratio (ratio of length to height) of the channel is 8. The computational domain was decomposed into 6 blocks and simulated using 6 cores of a computer.

Fig. 7 shows comparison of computational time for various parallel LBM codes for simulation of laminar forced convection in a 2D channel for grid sizes 201×1600 and 420×3200 for 1000 iterations with serial LBM code. The serial RLBM code takes 267 s for 1000 iterations. Among parallel codes, Matlabpool RLBM code takes 131 s respectively, i.e., roughly half the time required by serial RLBM code. While pMatlab and GPU RLBM code take 150 and 220 s respectively. Performance of parallel LBM codes are further tested by simulating fluid flow and heat transfer on another denser 2D grid, 421×3200. The Serial RLBM code takes 1500 s for 1000 iterations. Among parallel codes, Matlabpool RLBM code takes 870 s respectively, i.e., roughly half the time required by serial RLBM code. While pMatlab and GPU RLBM code take 880 and 1480 s respectively. This clearly reflects, that in any case Matlabpool parallel code takes less processing time for a given problem and a mesh size.

In parallel LBM (matlabpool and pMatlab) simulations, after every iteration, message containing particle distribution functions of the boundaries is passed to neighboring core or block. Message passing can be achieved in many ways. For instance, *MPI-SEND (Send-data*, *to-address*, *tag)* will send Send-data to *to-address* with a *tag*, *MPI-REVCIEVE(Rec-data*, *from-address*, *tag)* will receive *Rec-data* from *from-address* with a specified *tag*. This type of message passing in some cases will end up in dead-lock situations. Deadlock is a situation in message passing where a core or process will wait to receive a data from other process or cores indefinitely. It should be noted that when pMatlab is used for parallel simulation with 7 or more cores, message passing calls end in deadlock situation. The dead-lock situation can be avoided by using *MPI-*

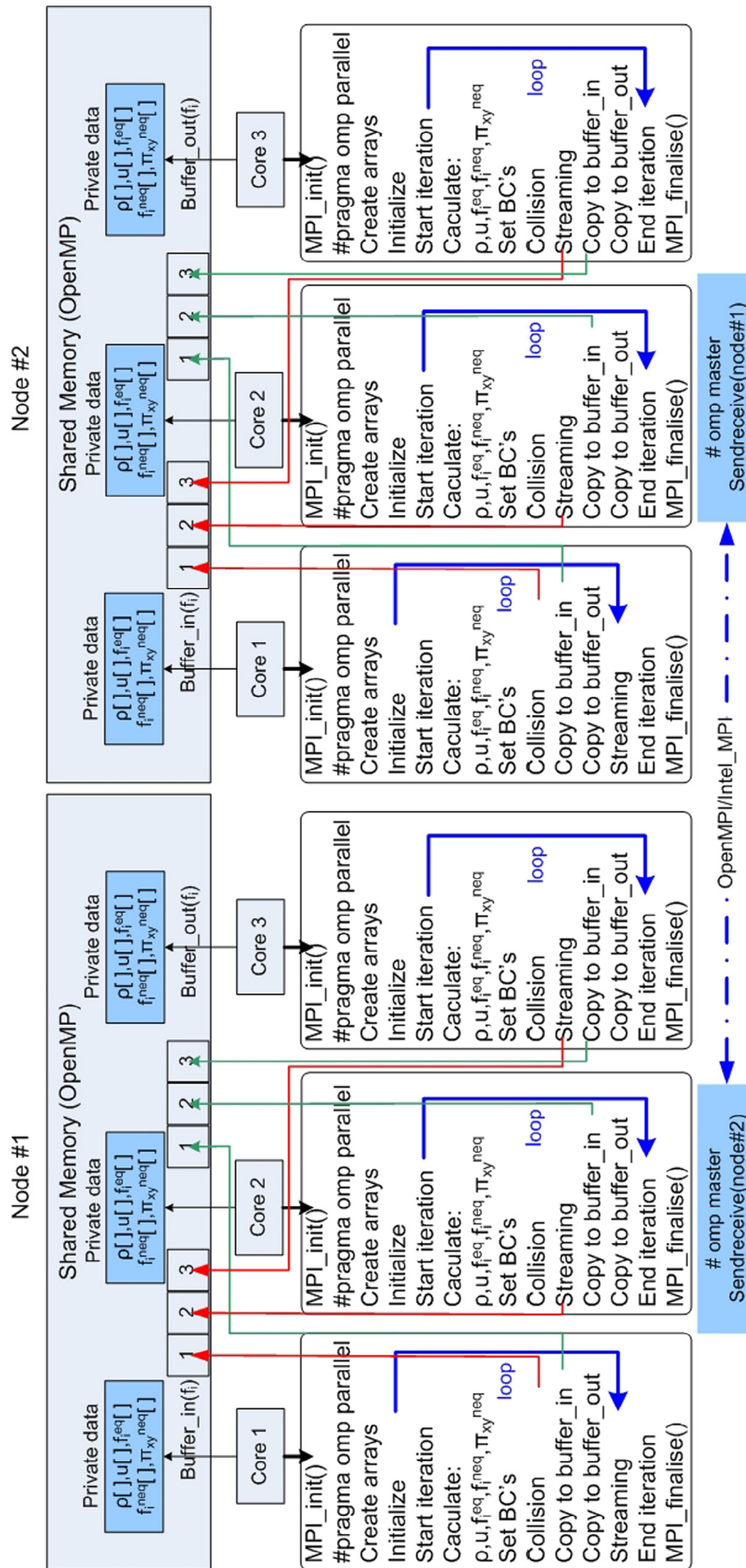

Fig. 5. Multi-core multi-node parallel LBM simulation algorithm on a HPC cluster using OpenMP and Intel-MPI/OpenMPI.

SENDRECIEVE() message passing calls, which make sure that message is sent and received from the same specified address or process simultaneously. This type of MPI call is available in Matlab library but not in pMatlab library. However, later on serial and parallel Matlab codes are rewritten in C language and compiled and simulated using GCC compiler for further enhancement in

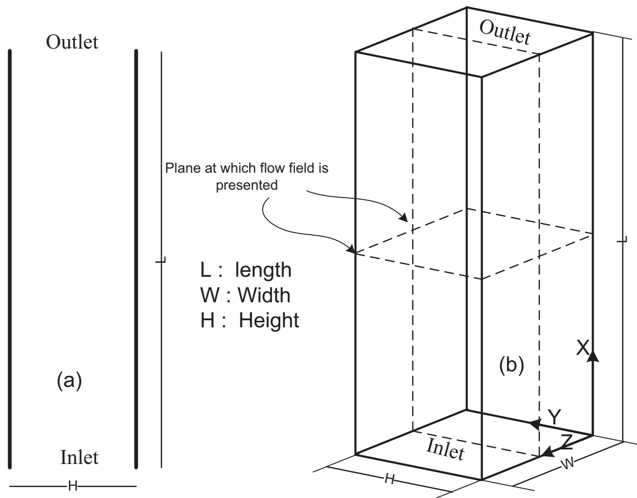

Fig. 6. Schematics of 2D and 3D channel.

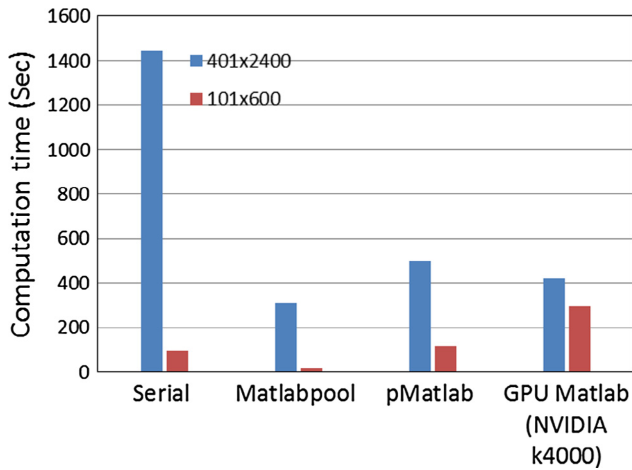

Fig. 7. Computational time (for 1000 iterations) for various types of LBM codes. Grid sizes 201 × 1600 and 420 × 3200.

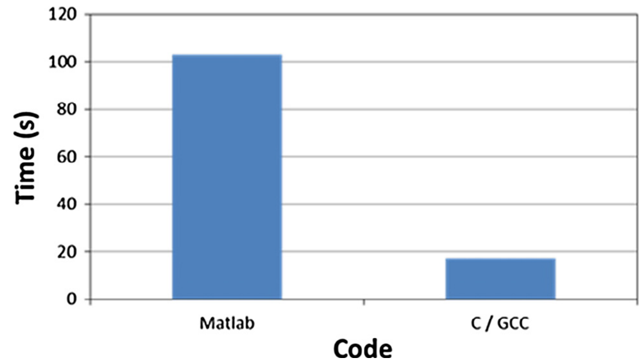

Fig. 8. Comparison of computational time (for 40000 iterations) for Matlab and C/GCC codes. Grid size 41 × 120, Re = 15.

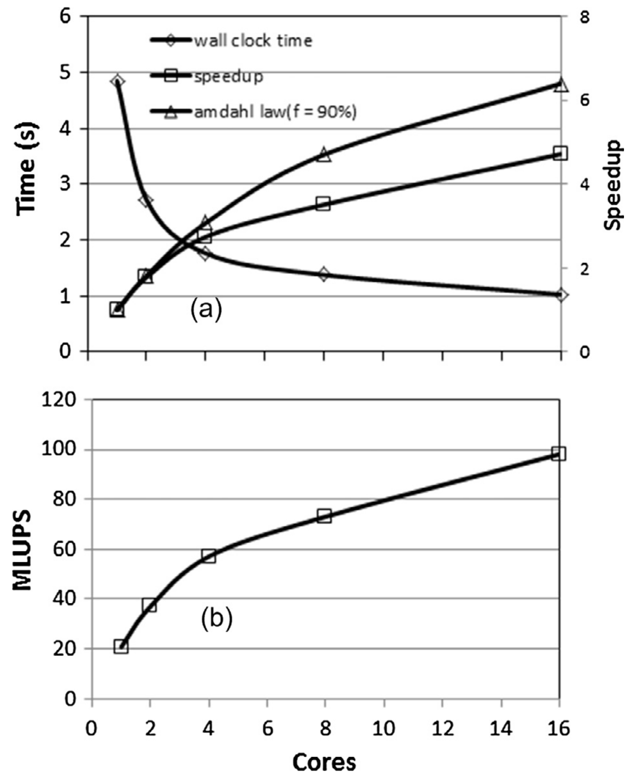

Fig. 9. Elapsed time, Speedup and MLUPS for parallel simulation using OpenMP on single node. (a) Speedup, (b) MLUPS.

the performance of the LBM codes. For a given 2D channel fluid flow problem, comparison of simulation time for Matlab and C code is presented in Fig. 8. The simulations are conducted on a desktop computer with dual core, core i5, 4 GB RAM. It can be seen that there is an appreciable gain (speedup 5X appx.) in performance by migrating from Matlab to C coding.

OpenMP compiler directive were used in the C LBM code to achieve parallelism. To test the performance of the parallel LBM code, simulations are conducted on a workstation equipped with 8 cores Zeon dual processor with 48 GB RAM and 3.2 GHz speed. As a test case, turbulent fluid flow and heat transfer is considered in a periodically fully developed 3D channel with D3Q19 and D3Q6 lattice topologies. Reynolds number based on friction velocity is 180 and constant temperature boundary condition is assumed. Fig. 9 shows elasped time, Speedup and MLUPS obtained for simulation of turbulent fluid flow and heat transfer by parallel LBM codes using OpenMP libraries. Time shown in the Figure is the elasped time per iteration, which is averaged over 1000 iterations. It can be seen that Speedup and MLUPS increases linearly with increase in OpenMP threads, up to 8 threads. There after there is no appreciable increase in performance with increase in parallel threads. It is worth to mention here that computations were also conducted for different grid sizes such as 50 × 71 × 71, 25 × 71 × 71, 6 × 71 × 71 and 100 × 71 × 71, but similar behavior in performance was noticed. Based on this observations, to get linearly scaled performance on a HPC cluster, OpenMP threads in each MPI thread of a node is restricted to 7–10 in all parallel simulations.

For a typical turbulent fluid flow and heat transfer in a 3D channel, parallel computations using OpenMP on a workstation will take a week time for 10^6 iterations for a moderate lattice grid size. It is inevitable to run the LBM code on a multinode cluster computer such as HPC for further enhancing the performance of the parallel code. Fig. 10a shows computational time per iteration for simulation of turbulent fluid flow and heat transfer by parallel LBM codes using OpenMP and OpenMPI libraries. To be in line with the literature [15] and for sake of comparison, Reynolds number is kept constant at 5328 and computational domain was discretised into 91 × 181 × 1080 uniform lattice nodes. Simulations are conducted for 1000 iteration and average time per iteration is

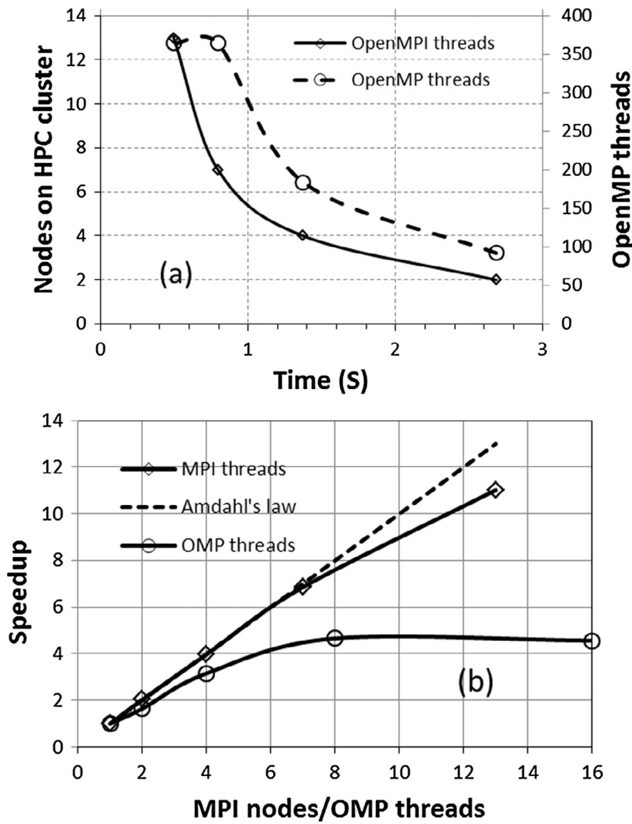

Fig. 10. Performance of parallel simulations on a HPC cluster. Grid size 91 × 181 × 1080, D3Q19 and D3Q6. (a) Computation time, (b) Speedup.

calculated. Simulations are conducted on HPC cluster with 32 nodes, 3.2 GHz dual processors, 40 GB RAM on each node. The HPC cluster nodes are inter-connected with High speed Infi-band network with LINUX operating system. It takes 2678 s for 1000 iterations using 2 nodes with 4 OpenMPI threads, with block size of 12 × 91 × 181, 23 OpenMP threads with two process on each node. Computational time decreases to 1345 s when 4 nodes with 8 OpenMPI threads, with block size of 6 × 91 × 181, 23 OpenMP threads with two process on each node are used. Computational time further decreases to 450 s when 13 nodes with 26 OpenMPI threads, with block size of 3 × 91 × 181, 14 OpenMP threads with two process on each node are used. It can be noticed that the performance increases linearly with increase in computational nodes up to 7 nodes, and there after performance gain is relatively lower. It is likely that further increase in computational nodes for parallel simulation will not help much in increasing the performance appreciably. It is worth to mention here that the parallel codes not only simulates fluid flow but also heat transfer in the 3D channel (91 × 181 × 1080 Grid × 19 + 6 lattice velocities = 444717000 data-size). The computational time per iteration using 13 nodes is less than 0.5 s which is more or less equal to the computational time for simulation of turbulent fluid flow only (91 × 181 × 1080 Grid × 19 lattice velocities = 337984920 data-size) reported by Dustin et al. [15]. In other words, the present turbulent LBM code is faster than the turbulent LBM code developed in Ref. [15]. Moreover, it is should be noted that the present code is hybrid code, that uses features of OpenMP and OpenMPI libraries for parallel simulation on shared and distributed memory architecture, while the code developed in Ref. [15] used features of OpenMP for parallel simulation on shared memory architecture with 256 processors on a single computer system. As explained earlier in the algorithm (Fig. 3), parallel simulations with OpenMP on shared memory computer system do not need message passing between parallel blocks. In other words, if the present code runs on a single computer with 256 processors on a shared memory architecture, the computation speed would be further higher. Fig. 10b shows gain in computational speed with increase in number of nodes on the HPC cluster. It can be seen that the computationl speed of the code (OpenMP & OpenMPI threads) scales almost linearly with number of nodes up to 7 node and thereafter performance degrades. whereas the computationl speed of the code (OpenMP threads) scales almost linearly with number of nodes up to 4 threads and thereafter performance degrades.

Fig. 11 shows averaged velocity field of turbulent fluid flow by parallel LBM code using OpenMP and OpenMPI libraries. The computation were conducted using 12 nodes with 24 OpenMPI threads. Seven OpenMP threads on each OpenMPI thread with

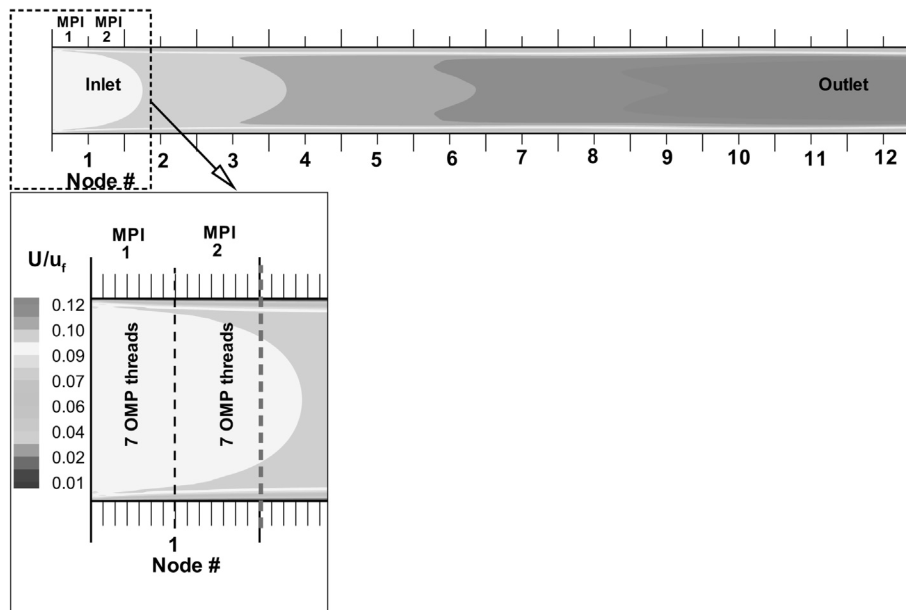

Fig. 11. Averaged velocity contours at mid-plane of a 3D channel. Re = 5328 ($Re_\tau = 180$).

block size of $4 \times 71 \times 71$ is used. The flow field is continuous along the channel and it represents a typical developing turbulent flow.

4. Summary

Incompressible laminar convection in 2D and 3D channels were simulated using LBGK method applying five different parallel libraries namely, Matlabpool, pMatlab, GPU Matlab (Naive), Openmp and OpenMP+OpenMPI. Lattice types D2Q9 and D2Q19 were considered for solving fluid flow in 2D and 3D channels, respectively and lattice types D2Q5 and D2Q6 were considered for solving heat transfer in 2D and 3D channels, respectively. Domain decomposition method was adopted for parallelizing uniform lattice grids. The performance of parallel LBM codes was compared with serial LBM code. Results showed that, for a given LBM simulation, pMatlab and Parallel toolbox perform almost equally well for small to moderate data array size, while for bigger data array size GPU performs well. For a given problem, simulation using C language with Openmp libraries were found to be faster than simulation with Matlab libraries. Further improvement in performance was found using C language and OpenMP+OpenMPI library on a HPC cluster with 32 nodes. The computational time for simulation of turbulent fluid flow and heat transfer ($91 \times 181 \times 1080$ Grid $\times 19 + 6$ lattice velocities = 444717000 data-size) per iteration using 13 nodes was found to be less than 0.5 s. The present turbulent LBM code is faster (35 %) than the turbulent LBM code developed in Ref. [15]. Turbulent fluid flow and heat transfer in 3D channels can be economically (in terms of computational time) simulated using LBM when used with hybrid parallel techniques such as OpenMP + Intel-MPI.

Conflict of interest

None.

References

[1] U. Frisch, B. Hasslacher, Y. Pomeau, Lattice-gas automata for the Navier-Stokes equation, Phys. Rev. Lett. 56 (1986) 1505.

[2] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems., Phys. Rev. 94 (1954) 511.

[3] Y. Huang, Q. Chen, A numerical model for transient simulation of porous wicked heat pipes by lattice Boltzmann method, Int. J. Heat Mass Transf. 105 (2017) 270–278.

[4] L. Jahanshaloo, N.A.C. Sidik, A. Fazeli, HA MP. An overview of boundary implementation in lattice Boltzmann method for computational heat and mass transfer, Int. Commun. Heat Mass Transf. 78 (2016) 1–12.

[5] L. Jahanshaloo, N.C. Sidik, S. Salimi, Numerical simulation of high Reynolds number flow in lid-driven cavity using multi-relaxation time Lattice Boltzmann Method, J. Adv. Res. Fluid Mech. Therm. Sci. 24 (2016) 12–21.

[6] S. Chapman, T.G. Cowling, The Mathematical Theory of Non-uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases, Cambridge University Press, 1970.

[7] X. He, L.-S. Luo, Theory of the lattice Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation, Phys. Rev. E 56 (1997) 6811.

[8] N. Latifiyan, M. Farhadzadeh, P. Hanafizadeh, M.H. Rahimian, Numerical study of droplet evaporation in contact with hot porous surface using lattice Boltzmann method, Int. Commun. Heat Mass Transf. 71 (2016) 56–74.

[9] M. Basha, C.N. Azwadi, Numerical study on the effect of inclination angles on natural convection in entrance region using regularised lattice Boltzmann BGK, J. Adv. Res. Fluid Mech. Therm. Sci. 10 (2015) 11–26.

[10] Z. Guo, B. Shi, C. Zheng, A coupled lattice BGK model for the Boussinesq equations, Int. J. Numer. Meth. Fluids 39 (2002) 325–342.

[11] C. Nor Azwadi, T. Tanahashi, Three-dimensional thermal lattice Boltzmann simulation of natural convection in a cubic cavity, Int. J. Mod. Phys. B 21 (2007) 87–96.

[12] M. Basha, C.S. Nor Azwadi, Regularized lattice Boltzmann simulation of laminar mixed convection in the entrance region of 2-D channels, Numer. Heat Transf. Part A: Appl. 63 (2013) 867–878.

[13] H. Amirshaghaghi, M. Rahimian, H. Safari, Application of a two phase lattice Boltzmann model in simulation of free surface jet impingement heat transfer, Int. Commun. Heat Mass Transf. 75 (2016) 282–294.

[14] N.A.C. Sidik, S.A. Razali, Various speed ratios of two-sided lid-driven cavity flow using lattice Boltzmann method, J. Adv. Res. Fluid Mech. Therm. Sci. 1 (2014) 11–18.

[15] N. Satofuka, T. Nishioka, Parallelization of lattice Boltzmann method for incompressible flow computations, Comput. Mech. 23 (1999) 164–171.

[16] O. Filippova, D. Hänel, Grid refinement for lattice-BGK models, J. Comput. Phys. 147 (1998) 219–228.

[17] J. Derksen, H.E. Van den Akker, Large eddy simulations on the flow driven by a Rushton turbine, AIChE J. 45 (1999) 209–221.

[18] D.M. Cherba, Performance analysis of a parallel implementation of the lattice Boltzmann method for computational, Fluid Dyn. (2002).

[19] K. Stratford, I. Pagonabarraga, Parallel simulation of particle suspensions with the lattice Boltzmann method, Comput. Math. Appl. 55 (2008) 1585–1593.

[20] C. Körner, T. Pohl, U. Rüde, N. Thürey, T. Zeiser, Parallel lattice Boltzmann methods for CFD applications, in: Numerical Solution of Partial Differential Equations on Parallel Computers, Springer, 2006, pp. 439–466.

[21] C. Schepke, N. Maillard, P.O. Navaux, Parallel lattice Boltzmann method with blocked partitioning, Int. J. Parall. Program. 37 (2009) 593–611.

[22] D. Bespalko, A. Pollard, M. 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, Springer, 2010, pp. 1–19.

[23] F. Schornbaum, U. Rüde, Massively parallel algorithms for the lattice boltzmann method on nonuniform grids, SIAM J. Sci. Comput. 38 (2016) C96–C126.

[24] X. Wang, Y. Shangguan, N. Onodera, H. Kobayashi, T. Aoki, Direct numerical simulation and large eddy simulation on a turbulent wall-bounded flow using lattice Boltzmann method and multiple GPUs, Math. Probl. Eng. 2014 (2014).

[25] V. Kumar, L. Hendren, MiX10: Compiling MATLAB to X10 for high performance. ACM SIGPLAN Notices, ACM, 2014, pp. 617–636.

[26] J. Kepner, Parallel MATLAB for Multicore and Multinode Computers, SIAM, 2009.

[27] J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Straight velocity boundaries in the lattice Boltzmann method, Phys. Rev. E 77 (2008) 056703.

[28] H. Yu, S.S. Girimaji, L.-S. Luo, DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method, J. Comput. Phys. 209 (2005) 599–616.

[29] M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Comparison of different propagation steps for lattice Boltzmann methods, Comput. Math. Appl. 65 (2013) 924–935.