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.

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/

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 × 181 × 1080 × 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.

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