PARALLELIZATION OF LATTICE BOLTZMANN METHOD FOR CFD USING MESSAGE PASSING INTERFACE

by

Shazia BASHIR[a*], Anila USMAN[b], Yasir MUMTAZ[c], Khaled H. MAHMOUD[d], Abdullah S. A. ALSUBAIE[d], Muhammad BASHIR[a], Farkhanda AFZAL[e], and Mustafa INC[f,g*]

[a]Department of Physics and Applied Mathematics and Center for Mathematical Sciences, Pakistan Institute of Engineering and Applied Sciences (PIEAS), P. O. Nilore, Islamabad, Pakistan
[b]Department of Computer and Information Sciences, PIEAS, P. O. Nilore, Islamabad, Pakistan
[c]Department of Chemical Engineering, PIEAS, P. O. Nilore, Islamabad, Pakistan
[d]Department of Physics, College of Khurma University College, Taif University, Taif, Saudi Arabia
[e]Department of Humanities and Basic Sciences, MCS, National University of Sciences and Technology, Islamabad, Pakistan
[f]Department of Mathematics, Firat University, Elazig, Turkey
[g]Department of Medical Research, China Medical University, Taichung, Taiwan

Original scientific paper
https://doi.org/10.2298/TSCI22S1211B

The lattice Boltzmann method has become a promising numerical technique and is now being considered as an alternative to the conventional CFD methods owing a possibility to simulate more complex geometries at relatively low computational costs. The simulations of complex and very fine resolution computational domains in CFD is challenging due to the lack of memory resources and long processing times, therefore parallel computing is being considered as a promising way to cope with this ever increasing mission of computational power. In this work parallelization of 2-D lattice Boltzmann model based CFD code using message passing interface was performed to increase speedup factor for parallel computing. Lattice Boltzmann approach based CFD code of a benchmark Lid driven cavity flow problem was parallelized using different message passing interface subroutines and made to run on a cluster system of processors. The time and speed up factor for execution of the benchmark problem was investigated. The results showed that these message passing communications have little influence on the performance of the parallel lattice Boltzmann method.

Keywords: *lattice Boltzmann method, benchmark problem, message passing interface*

Introduction

A considerable work has been done to develop the lattice Boltzmann techniques which can simulate flows including multiphase flows even in porous media and complex geometries and for high values of Reynolds number [1-4]. Most of the work was done on

*Corresponding authors, e-mail: shazia@pieas.edu.pk, minc@firat.edu.tr

understanding of lattice Boltzmann method (LBM) with different boundary conditions such as bounce back boundary condition, inlet boundary conditions, pressure boundary conditions, *etc*. To simulate the complex and large systems, large resolution is needed to get the adequate results but in this case, computational power was not enough in the past few decades. Therefore, magnificent work was done on high performance computing including parallel computing. The LBM employs discretized particle distribution functions based on a microscopic fluid physics to follow the thermal and hydrodynamic flow field in contrast to the conventional CFD methods that solve the Navier-Stokes equations. All main areas of transport phenomena including momentum, energy and mass transport can be simulated with LBM. In lattice Boltzmann model approach, the computational domain or the geometry of the system is discretized same as meshing in conventional CFD and so called particles are assumed at each node in the computational grid. Particles are allowed to stream in specific restricted directions according to different models that have been developed basically on the number of physical dimensions in the system such as 1-D, 2-D or 3-D systems. These particles then are allowed to collide with other particles during streaming and then stream again after collision in the same fashion. The time difference between two successive collisions or streaming steps is called as relaxation time which is an important parameter in LBM approach. These streaming and collision are executed in such a way that the coarse graining of particles distribution will result in the recovery of weakly compressible or almost incompressible Navier-Stokes equations [5, 6].

In general, running simulations on large systems is not practical due to the lack of memory resources and long processing times. Due to these limitations and the fact that the LBM needs only nearest neighboring information, LBM is the ideal candidate for parallel computing. In parallelization, a sequential code is converted into a threaded form. Thread is the smallest independent simulation code that run typically on its own processor and own block of data which in turn minimizes the time as well as memory required to run the simulation. Therefore, multiple processors are made to run simultaneously. The time for parallelization of a code is directly proportional to the fraction of the code that can be parallelized. Parallelization of LBM can be done using message passing interface (MPI) and OpenMP. The MPI is basically distributed memory model based and executed on distributed networks through proper send and receive of messages while Open MP is shared memory model based and used on multi-core processors and is relatively easier to program and debug as compared to MPI based programs [7].

This research aims to implement LBM for parallelization of the CFD benchmark problem using MPI and run on multiple processors cluster computing system. The 2-D D2Q9 model was used as it is capable of recovering Navier-Stokes equations at macroscopic level. The domain decomposition, data partition, and parallelism of the LBM code with various messaging passing communications using the MPI library have been investigated. The time required for execution and speed up factor was compared for serial and parallel tasks for execution of the program.

Lattice Boltzmann method

The LBM is an explicit technique to solve fluid-flow problems. These equations are capable of solving momentum, energy, and mass transport phenomena. The Boltzmann equation (BE) contains the single particle distribution function as its integral part as given by:

$$\frac{\partial f}{\partial t} + \frac{P}{m}\nabla f + F\frac{\partial f}{\partial P} = \frac{1}{\tau}\left(f^{\mathrm{eq}} - f\right) \tag{1}$$

where *f* is the particle distribution function. It can be of any macroscopic physical property of system like temperature, density, *etc*. Consider *f* to be the density distribution function for the time being. Then the first term in eq. (1) is the time dependent term which tells the change of density distribution function as a function of time. Second term is the change in density distribution function due to change of velocity in three directions (*x*, *y*, *z*). Third term in equation explains the force due to which the distribution function can change. Fourth term on the right hand side of equation explains the change in distribution function as a function of time due to the collision between two particles. Considering *f* to be the density distribution function, there are seven variables in eq. (1), three displacement variables (*x*, *y*, *z*), three velocity variables (V_x, V_y, V_z), and time. The BE when discretized is called as lattice Boltzmann equation. Lattice Boltzmann equation can be split into two steps: collision and streaming. In collision step, particles on the nodes collide and velocities are changed. In streaming, particles move from their nodes to the neighbor nodes according to the restricted velocity directions as given for different models [6]. Collision and streaming are the major stages in advancement of LBM simulations on discretized domains. After this collision and streaming, each cell information is updated waiting for the next update in next time step. This is how information of each node is updated after successive collision and streaming steps until a steady state is reached.

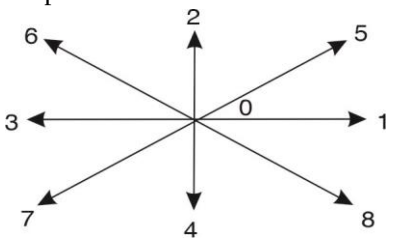

Figure 1. The D2Q9 model

This model is a 2-D model having nine velocities. The central particles are at rest. Eight directions are for eight velocity vectors. Lattice width and height is 1 lattice unit. The user can set lattice unit according to the demand of the problem in hand, and other unit is time step (ts) it is the time in which one collision and one propagation occurs, fig. 1. Therefore, the lattice speed, *c*, is defined as lattice units per time step (lu/ts). Vector (1 to 4) has velocity magnitude of 1 lu/ts. Vectors (5 to 8) have velocity magnitude of $\sqrt{2}$ lu/ts.

Following assumptions convert BE of D2 Q9 Model to LBE.

- Single relaxation time.
- Velocity discretization i.e. using finite set of velocities.
- Particles can move only along eight directions.
- Modeling of fluid by many cells of same type.
- Update of cells at each time step.

Equilibrium density distribution function:

$$f_i^{\mathrm{eq}} = \rho w_i \left[1 + \frac{3}{c^2} e_i u + \frac{9}{2c^4} (e_i u)^2 - \frac{3}{2c^2} uu \right] \tag{2}$$

where ρ is the localized density, w_i – the weighting fraction specific for each velocity vector, $c = \Delta x/\Delta t$ (value of c is usually 1 as both Δx and Δt are kept mostly unity or very close to unity often), and u – the macroscopic velocity. All these values are easy to calculate and complexity of collision term was eliminated by BGK approximation.

Macroscopic density:

$$\rho = \sum_{i=0}^{8} f_i \tag{3}$$

Macroscopic density at any point is the sum of density distribution function on that node.

Relaxation parameter:

$$\Omega = \frac{1}{3\nu + 0.5} \tag{4}$$

Density relaxation parameter is calculated by using the macroscopic viscosity of the fluid.

Parallel computation

In parallelization, the serial code developed is decomposed into different data parts or blocks and then those data blocks are assigned to number of processors which process those data blocks simultaneously meaning that multiple number of processors is made to run at the same time to do the same job. We need parallel computing mostly for large systems because computational cost is too high for a single processor [8]. Some processes are relatively simple to code and compute mainly due to their boundary conditions but when we have very fine resolution of grid or very fine discretization, computation cost exceeds too much and then parallel computing for those processes comes into play for which MPI is the most favorable option. Processes are parallelized to complete different tasks in the shortest amount of time. Small execution time can only be achieved when overheads in parallel execution of a program are minimized. It also depends on the decomposition of a problem in different fashions. For a given decomposition technique, overheads are mainly due to the time elapsed in inter-process communications and time that some processors spent being idle even when there is work to do like the case of deadlocks. Sometimes a processor can be free or idle before the overall computation is done or sometimes all other processes can be idle waiting for the job done by some processor who has lot of work to do as compared to all other processors due to the uneven load distribution for the completion of the whole job. Both the communication and idling are a strong function of mapping of the processors in virtual grids. A good mapping scheme must reduce the interaction or communication time of different processors as well as reduce the total amount of time in which some processors are idle while all other are engaged in some task executions.

The speedup is the terminology in parallel computing which is defined as a ratio of serial to parallel execution times has also been studied in our case:

$$\text{Speed up} = \frac{T_1}{T_n}$$

where T_1 is the serial execution time *i.e.* on a single processor and T_n – the parallel execution time *i.e.* on multiple processors.

Parallelization of LBM using MPI

The main parameter to be calculated in parallelization of LBM for lid driven cavity flow is the distribution function *f*. As this is a 2-D method employing D2Q9 model, distribution function has nine discrete values which would be calculated in each collision step and transmitted in each streaming step for each iteration. Following points enlist the procedure of implementation of code for parallel computation:

- Overall computation domain containing 1000 × 1000 grid, (length × width) was decomposed in *n*-sets of ($\text{length}_{\text{sub}} \times \text{width}_{\text{sub}}$). Where *n* is the number of procesors.
- Sub domains were tried to be kept as square as possible which in turn maximizes the ratio of computation to communication time because of the equal load distribution among different processors.

- This also reduced the idling of processors due to unequal load distribution because each processor was assigned with perfectly equal square data block to be computed.
- At each time step, each processor updated the fluid nodes(distribution functions) in its own sub domain.
- The $MPI_{REQUEST}$ along with $MPI_{STARTALL}$, and $MPI_{WAITALL}$ were used to send results from slaves to master processor. The $MPI_{STARTALL}$ initialized the sending operation of all slave processors at the same time which reduced the idling of processors which would have been present if different processors send their output to master processor at different times. On the other hand, $MPI_{WAITALL}$ held each processor until a specific operation was done by each processor in the communicator. This reduced the randomized distribution of calls and data.
- Bounce-back boundary condition was applied for left, right and bottom wall whereas moving lid boundary condition was applied for top lid.
- Macroscopic density and velocity were calculated as:

$$\sum_{\sigma}\sum_{i} f_{\sigma i}^{(0)} = \rho, \quad \sum\sum f_{\sigma i}^{(0)} e_{\sigma i} = \rho u \tag{5}$$

- Storage of arrays in both FORTRAN and C is different being stored as columns and rows respectively. Even for higher dimensional arrays, the most varying index is the first one and it can be used to access the large memory stored in computer memory. Therefore, this index was used to access the arrays stored.
- Continuous memory access was used as it caused less cache miss than other cases in which simple jump access is used which is not efficient enough to reduce the execution time of a problem.
- Blocking send and receive means that once a process sends a message to other processor, the sending at previous processor is blocked until the receiver has received the process and vice versa. As is our case, 8 distribution functions values at each node are updated for each time step and they are further sent and received between neighbor processors. Therefore, Ordered send and received were used i-e MPI_ISEND and MPI_IRECV. This ordered send and receive reduced the time for communication and even in some cases the deadlocks.
- This was done by pairing the send and receive in such a fashion that when one processor is sending its message to other, the other processor will receive the message first matching that send and after that the other processor sent the message of its own to the other processors and so on.
- Results were tested by varying different parameters like overall computation domain size that is number of nodes in *X* and *Y*-dimensions, number of time steps and number of processors, *etc*.

Results and discussion

In this paper, parallel implementation of LBM to a bench mark CFD problem of lid driven cavity flow using MPI has been studied. The lid of cavity is moving in a positive horizontal direction with the liquid present in tank. The schematic representation is shown in fig. 2. The LBM code has been implemented in LINUX operating system and run on single processor for noticing the execution time in case of serial code and on multiple processors up to 32 processors in case of parallel codes.

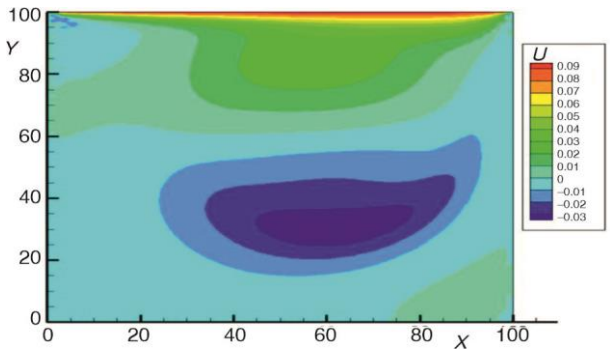

Figure 2. Contour of velocity in *x*-direction

The cluster system used in PIEAS has the following specifications:

- Two Intel Xeon pro 2 GHz with 4 MB cache, 4 Cores, and 32 GB memory each.
- Six Intel core i5 pro 2.67 GHz with 8 MB cache, 4 Cores, and 4 GB memory each.

Therefore, overall there are total 32 cores that have been used for parallel computing.

The computational parameters in this method are:

- Grid Size: $m \times n$ =1000 × 1000, where m is the number of nodes in y-axis and n is the number of nodes in x-axis.
- The total number of time steps = 150000.
- Bounce back boundary condition for left, right and bottom wall whereas, moving lid boundary condition for top lid.
- Time for execution on single Intel Xeon 2 GHz with 4 MB Cache, 4 Cores, and 32 GB memory = 29132.34 sec.

The top lid of the cavity is being moved in horizontal direction while viscous liquid is present in tank. When lid moves, the liquid which is in contact with the wall causes and induced motion in bulk of the liquid. Serial execution of this method gives following results which have been produced in fig. 2 using Tecplot for a coarse grid just for a clear view of different flow regimes. Where X represent the number of nodes in the horizontal direction and Y represent the number of nodes in the vertical direction of cavity. The red color region shows strong components of velocity while regions with light green color show a very little movement of fluid and the regions indicated with blue color show negative velocity meaning that fluid is not being moved at all. Green colored regions show the dead zones where there is no effect of movement of lid.

For a grid size of 1000 × 1000 and 150000 time steps, parallelization results up to 32 processors are presented herein. The same problem was executed on different number of processors and the results obtained are presented in fig. 3. The results of number of processors versus time of execution and speedup can be shown graphically in fig. 4 and 5, respectively.

It is evident from fig. 4 that the execution time is decreased as the number of processors was increased. It is obvious that initially the slope is near to that of linearity showing near to ideal results of time decrease versus number of processors but as we increase number of processors in communicator, the inclination of slope starts to decrease showing a trend slightly away from linearity due to the domination of communication time slowly.

The fig. 5 shows that, initially the speedup is in exact accordance of Ideal speedup shown as straight line but it starts to divert when the number of processors are increased beyond four. This is mainly due to the increase of communication time among different proces-

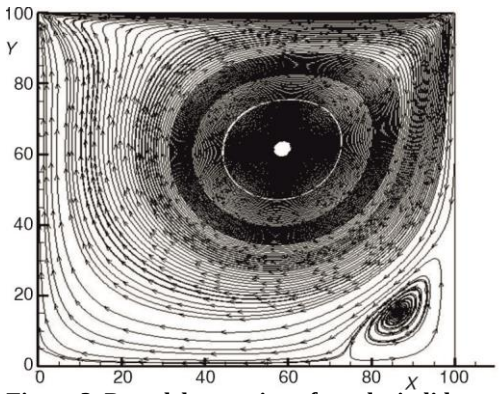

Figure 3. Parralel execution of results in lid driven cavity (stream traces of axial velocity)

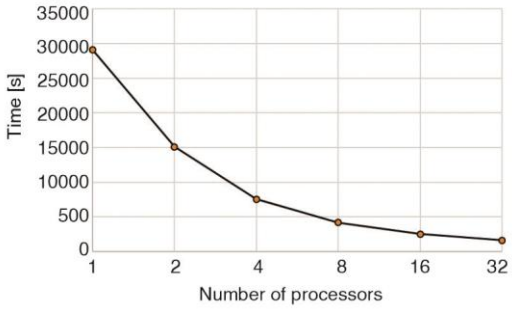

Figure 4. Execution time *vs*. number of processors

sors in overall communicator in MPI environment. As a benchmark, the results have been compared with that of [9] and found to corroborate. The trends observed in both cases are almost the same. When we increase number of processors up to four, the efficiency of parallel computing is close to unity and staying almost constant. The efficiency of parallel computing increases as the fraction of a problem that can be parallelized increases. Graphically efficiency can be represented in fig. 6.

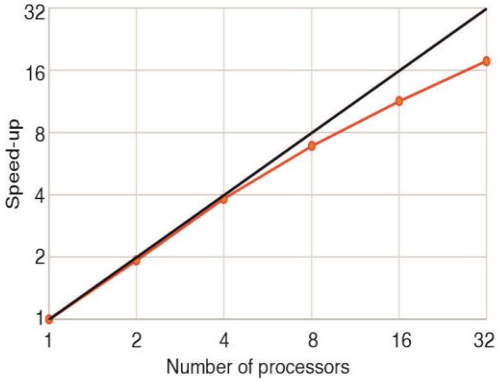

Figure 5. Speed up *vs*. number of processors

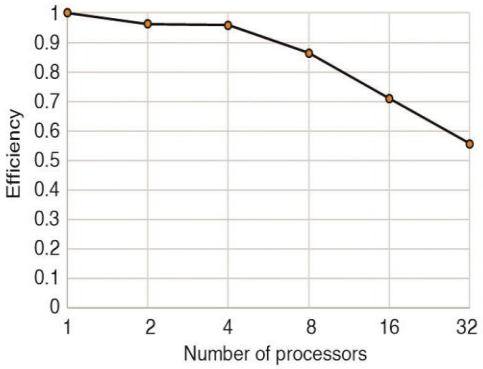

Figure 6. Efficiency *vs*. number of processors

It can be seen that when we increase the number of processors up to four, the efficiency of parallel computing is close to unity and keep on almost constant.

Conclusion

In this paper parallelization of lattice Boltzmann D2Q9 model has been implemented using MPI. A 2-D lid driven cavity flow was simulated when lid was moving in positive x-direction with the liquid present in tank. The simulations were performed using both on serial and parallel computation. It was observed that the execution time in parallelization has been reduced near to ideal trends. Maximum parallelization efficiencies of 89% and 96% have been achieved. An efficiency and speedup of parallelized problem show a decreasing trend when numbers of processors are increased gradually which was mainly due to the gradual domination of communication time over computation time. It was inferred from outcomes of this

study that LBM is a remarkable tool for simulating fluid-flow problems when it is parallelized. This has potential for implementation in more complex CFD problems utilizing more number of processors.

Acknowledgment

The authors would like to acknowledge the financial support of Taif University Researchers Supporting Project number (TURSP-2020/162), Taif University, Taif, Saudi Arabia and the department of computer and information sciences of PIEAS for providing the parallel computational facility.

References

[1] Luo, L. S., *et al*., *Lattice Boltzmann Method for Computational Fluid Dynamics*, Encyclopedia of Aerospace Engineering, John Wiley & Sons, New York, USA, 2010
[2] Chopard, B., Dupuis, A., Lattice Boltzmann Models: An Efficient and Simple Approach to Complex Flow Problems, *Computer Physics Communications, 147* (2002), 1-2, pp. 509-515
[3] Mohammad, A. A. *Lattice Boltzmann Method*. 2nd ed., Springer*Verlag, London Ltd., 2019
[4] Chen, S., Doolen, G. D., Lattice Boltzmann Method for Fluid-Flows, *Annual Review of Fluid Mechanics, 30* (1998), 1, pp. 329-364
[5] Inamuro, T., Suzuki, K., *An Introduction to the Lattice Boltzmann Method*, Word Scientific, Singapure, 2021
[6] Cristea, A., Numerical Schemes for Lattice Boltzmann Models, *Romanian Reports in Physics*, *58* (2006), 3, pp. 319-324
[7] Zhang, F., Research on Parallel Computing Performance Visualization Based on MPI, *Proceedings*, 2nd Int. Conf. on Advanced Computer Control (ICACC), Shenyang, China, pp. 323-327, 2010
[8] Grama, A., *Introduction to Parallel Computing*, 2nd ed., Pearson Education, Singapore Pte. Ltd., 482, 2004
[9] Ni, J., *et al*., Parallelism of a Lattice Boltzmann Method (LBM) for Lid-Driven Cavity Flows, University Ilinois research booth, Supercomputing, Chicago, Ill., USA, 2003

Paper submitted: September 10, 2022
Paper revised: October 18, 2022
Paper accepted: October 24, 2022