Ninth International Conference on Computational Fluid Dynamics (ICCFD9), Istanbul, Turkey, July 11-15, 2016

ICCFD9-xxxx

Channel and Cavity Flow Analysis Using Lattice Boltzmann Solvers Parallelized with Coarray Fortran

Fadile Yudum Comez[1] & Nevsan Sengil[2]

[1]Department of Aerospace Engineering, Middle East Technical University, Ankara 06840, Turkey
[2] Faculty of Aeronautics and Astronautics, University of Turkish Aeronautical Association, Ankara 06790, Turkey

Corresponding author: yudum.comez@metu.edu.tr

Abstract: Lattice Boltzmann method (LBM) has some distinct advantages compared to the conventional computational fluid dynamics methods such as Navier-Stokes and Euler. Program developers take advantage of the LBM's simple algorithm and implement Lattice Boltzmann method successfully to single and multiphase flows in the complex geometries. The unique programming structure of LBM also makes the parallelization technique quite practical. In this study we focused on the parallelization of the LBM with Coarray Fortran (CAF) technique. There are some benefits of the CAF. First, CAF requires less line of codes. Next, understanding and maintaining the code developed by CAF is easier. Finally, parallelization efficiency of the LBM is relatively high because of the locality of the data. We also aimed to develop and run our LBM solver on a multi-core workstation. Using domain decomposition technique, the whole computational domain is further divided to sub-domains and each sub-domain is assigned to a different core for calculations. Data exchanges between border lattice points are realized with the coarray feature of the CAF. We decided to study on two leading test cases such as lid-driven cavity and channel flows to test our solver. Tests were realized with Intel Fortran XE compiler. In these tests, we used different number of lattice points and D2Q9 model. After validation tests of the parallel LBM solver, solution times and parallel efficiencies were calculated and plotted to show the parallel performance figures of the Coarray Fortran.

Keywords: Coarray, Fortran, cavity flow, channel flow, parallelization, CFD.

1 Introduction

Last two decades, some new methods have emerged to analyze fluid flows rather than conventional methods based on the continuum equations, namely, Navier-Stokes and Euler. Cellular automata (CA) method was developed to solve the Navier-Stokes equations indirectly [5].In CA method, fictional fluid particles propagate to neighbor lattice points according to their directional velocity vector in each iteration. After propagation occurred, collision step is executed. In this step, particles which are on the same lattice point interact in accordance with the collision model. But three important drawbacks of the CA method limited its application range [9]. First, CA method is computationally very expensive in high Reynolds number. Second, CA method is valid only in small Mach numbers. Finally, CA method accommodates high noise difficulty because of its statistical character. The difficulties with noisiness of CA systems are avoided by using a lattice Boltzmann approach. McNamara and Zanetti propose the earliest version of the LBE by averaging the Boolean occupation

numbers to resolve the noise problem [11]. Starting from the first introduction, LBM attracted the interest of the many researchers owing to its simple algorithm and programming structure. New developments extended the application range of the LBM method to include multiscale flows, complex solid-liquid interface, surface reactions in fluids colloidal solutions and turbulence [15].Simultaneously, additional studies were also carried out to increase the efficiency of the LBM solvers using sophisticated parallelization techniques. There are many studies in the literature to parallelize the lattice Boltzmann method in order to reduce the solution time. One early example is the parallelization of the LBM solver on the IBM SP1 scalable parallel computer system using MPL (Message Passing Library) and PVM (Parallel Virtual Machine) tools [14]. Later MPI (Message Passing Interface) superseded other libraries and became a prominent parallelization tool for the new LBM solvers [8]. Load balance and memory efficiency of the parallel LBM solvers were also examined in fluid flows through the porous media. In this work Fortran programming language and MPI is used [20].

LBM method is also parallelized with OpenMP method for the shared-memory architectures [1]. While MPI is a library consist of message passing routines, OpenMP is language extensions run on shared memory systems. Recently, fast growing GPU (Graphic Processing Unit) based computers attracted the attention of some LBM code developers inherently and a two-dimensional parallel D2Q9 LBM solver was run on the single-GPU computer [19]. Following, D3Q19 lid-driven cavity problem on the six NVIDIA Tesla C1060 GPUs working in parallel was analyzed [13]. In these last two studies, MPI libraries were employed. In another study, the application range of the LBM method was further extended to include the turbulent flows at high Reynolds numbers on GPU platforms using CUDA [10]. In addition to the conventional techniques such as MPI and OpenMP, OpenCL (Open Computing Language) and PGAS (Partitioned Global Address Space) paradigm were also implemented to parallelize the lattice Boltzmann method [18][6][16]. The PGAS model may be placed between the message-passing models such as MPI, and shared-memory models (OpenMP). Both models have different approach to exchange information between processes and memory. In MPI where isolated processes with isolated memories exchange messages and multiple threads can read and write a shared memory in OpenMP. When these are taken into account, PGAS is relatively simple for developing and debugging the parallel programs. Moreover, the design of the PGAS is compatible with both distributed and shared memory systems. In addition, the PGAS is powerful model helped to reach exascale computing performance which is described as a thousand peta flops or a quintillion, 10^{18}, floating point operations per second. Consequently, some of the well-known programming languages such as C, Fortran and Java incorporated this new model as extensions such as UPC (Unified Parallel C), Coarray Fortran (CAF) and Titanium respectively.

In the current study, CAF is used to parallelize our sequential lattice Boltzmann solver which is written with Fortran. We prefer the CAF for parallelization over the others, because first of all CAF requires less line of codes when compared with MPI. However, OpenMP parallelization almost goes head to head with CAF [4]. Additionally, understanding and maintaining the code developed by CAF is easier. Parallelization efficiency of the LBM is relatively high because of the locality of the data. Our parallel solver is also intended to run on the note-books and workstations which involve small number of cores. In this paper, the lid-driven cavity and channel flow test problems are sequentially solved and then the parallel solvers based on the Lattice Boltzmann method are implemented into the test problems. Results are compared and reported in terms of parallel efficiency and solution times.

2 Lattice Boltzmann Method

In this method, complicated Boltzmann equation is first simplified with BGK approximation and solved on a discretized mesh. In addition to the space, the velocity of the particles is also discretized $(\mathbf{c}_a)$. The number of the discretized velocities depends on the model chosen. There is only one unknown function which is particle distribution function (f_a) in the LBM. Distribution function

$f_a(\mathbf{x},t)$ is that a particle at time t is at position $\mathbf{x}$ and the probability of the particle moving with speed $(\mathbf{c}_a)$ being at position $\mathbf{x}$ at time t. Particle distribution function (f_a) changes with particle movements as shown below,

$$f_a(\mathbf{x}+\mathbf{c}_a\ \Delta t, t+\Delta t) = f_a(\mathbf{x},t) - \frac{f_a(\mathbf{x},t) - f_a^{eq}(\mathbf{x},t)}{\tau}$$

where Δt, $\mathbf{c}_a$, $\mathbf{c}_a\ \Delta t$, τ, and f_a^{eq} are time step, discrete particle velocities, displacement in position, single relaxation time, and equilibrium distribution function, respectively. Above equation consists of two parts namely, streaming and collision steps. The steps are represented by the left-hand and right-hand sides of this equation. In streaming step, f_a values move to the nearest lattice points. In collision step at mesoscopic scale, f_a values are forced to relax towards the equilibrium value. The time is covered between the actual and equilibrium configuration by giving the relaxation time constants. In simulating incompressible (e.g. water) flows the relaxation time can be taken constant. These steps repeat each other. Various lattice models are used for LBM. D2Q9 model is one of the most familiar model [17]**.** In this model D2 and Q9 represent the dimension of the geometry and the number of discrete velocities, respectively. The equilibrium distribution function in D2Q9 model is defined as shown below,

$$f_a^{eq} = w_a\ \rho(\mathbf{x})\left[1 + \frac{3(\mathbf{c}_a\cdot\mathbf{u})}{c^2} + \frac{9(\mathbf{c}_a\cdot\mathbf{u})^2}{2\,c^4} - \frac{3\,\mathbf{u}^2}{2\,c^2}\right].$$

Here w_a (weight values) are predetermined values and given as $w_0 = 4/9$ $w_1 = w_2 = w_3 = w_4 = 1/9$ and $w_5 = w_6 = w_7 = w_8 = 1/36$. is the propagation speed on the lattice. The discrete velocities $(\mathbf{c}_a)$ are given as $(0,0)$ for $\alpha = 0$, $c\left[\cos((a-1)\pi/2), \sin((a-1)\pi/2)\right]$ for $a = 1,2,3,4$ and $\sqrt{2}c\left[\cos((a-5)\pi/2+\pi/4), \sin((a-5)\pi/2+\pi/4)\right]$ for $a = 5,6,7,8.$

Once distribution function is known, various macroscopic properties such as density (ρ),velocity $(\mathbf{u})$,pressure (p) , and viscosity (ν) can be calculated through moment integrations as shown below,

$$\rho(\mathbf{x},t) = \int_{a=1}^{9} f_a$$

$$\mathbf{u}(\mathbf{x},t) = \frac{1}{\rho}\sum_{a=1}^{9} f_a\mathbf{c}_a$$

$$p(\mathbf{x},t) = c_s^2\ \rho$$

and

$$\nu(\mathbf{x},t) = \frac{(\tau-0.5)\,c^2\Delta t}{3}$$

The speed of sound is a lattice dependent quantity, which takes the value for D2Q9 model. In LBM different type of boundaries can be modeled such as periodic, bounce back, pressure (Dirichlet), and velocity (von Neumann). Periodic boundary conditions are the simplest boundary conditions. These boundaries can be treated as if they are attached to each other. For solid type boundaries, mostly simple "bounced-back" model is preferred. This model comes up to a no-slip boundary condition. In this model, incoming and reflected angles of the particles from the wall-type boundaries are equal.

For the pressure and velocity boundary conditions additional equations are defined to close the equation system [2]. Unknown distribution functions at the boundaries should be calculated from the given macroscopic values [21].

3 Coarray Fortran model

Coarray Fortran (CAF) started as an extension of Fortran 95/2003 for parallel processing in the 1990s [12]. Now Fortran 2008 includes CAF to ease the development of the parallel computer programs running on the many or multi processors environments. In this study, a multi processors environment is considered. Thus, dealing with shared memory systems, Intel Xeon E5506 @2 CPUs with the non-uniform memory access (NUMA) architecture is preferred for parallelization phenomena. 4 MB SmartCache memory is available on. In addition, each CPU has 4 number of cores. CAF is based on the partitioned global access space (PGAS) paradigm. It is developed to take the best parts of the shared and distributed memory architectures. In shared-memory model, all the processors share a global memory unit. Data locality is not a concern and any processor can access any location on the global memory. This system is good for the fast code development and loop-level parallelism. However, synchronization among processors and data races can be problematical. OpenMP is the leading model used in shared-memory architectures. In the distributed-memory model, all the processors use their own local address spaces. Data transfers are realized through the sending and receiving messages. Localization of the data speeds up the processor's execution time. However, message transfers used for transferring data can decrease the computational efficiency and difficult to debug. Shortly, PGAS is a relatively new parallel programming paradigm to be taken into account when developing parallel working programs with less effort without sacrificing the performance.
In PGAS, global memory address space is logically partitioned [3]. Therefore, some data is local while other data is global. Local data access speed is faster than the global data. Data transfers among the processors are realized through the global data. The CAF programming bases on three primary issues. The first two are how the code is parallelized in terms of work and data distribution. The third is the synchronization problem. For the work distribution, the CAF adopts the single program multiple data (SPMD) execution model. Same program is executed by each of the processors. In CAF, these copy-programs are called "images". The number of the images is determined by the programmer. This number can be equal or different from the number of physical processors. In CAF, the intrinsic function "num_images ()" points to the number of images determined by the programmer. In case of data distribution, the data locality is exploited by the CAF using co-dimensions shown by square bracket [n].The number "n" identifies the "image" to which data array belongs. The data distribution evolves that the local data are executed by each image and attainability of the local array is pointed by asterisks in square bracket [*]. A data array without a square bracket is an ordinary array and can be processed only with local "image". Remote "images" cannot access these local data arrays. In CAF, the second important intrinsic function is "this_image ()" that shows the identification of the local "image". PGAS data architecture and data access model are shown in Figure 1.

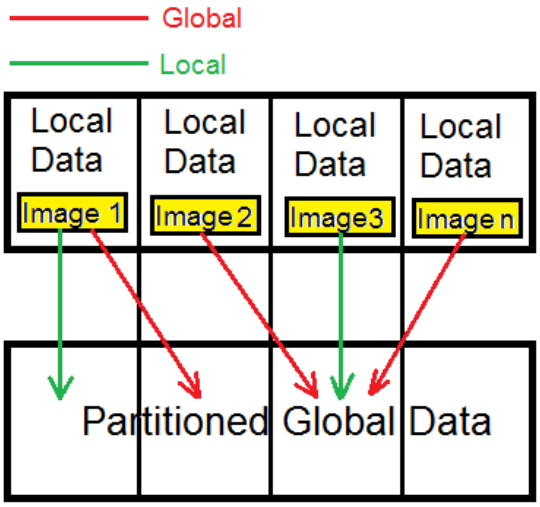


Figure 1 PGAS data distribution and access

Synchronization of the "images" has a great deal of importance for programming with the CAF. Using "sync all" intrinsic function inside the CAF, all the images are forced to wait until all of them finish their local or remote accesses.

3.1 Parallelization of the LBM solver

We used a small number of CAF syntaxes to develop a parallel LBM solver. Moreover, all the over head and communication issues are handled by the CAF complier. The small number of syntaxes simplified the program development and debugging otherwise a cumbersome job. The CAF syntaxes used in the parallel LBM solver are shown in Listing 1. The code fragment tells that an array-of-structures data layout.

List 1 CAF syntaxes used in parallel LBM solver.

```
! Global array declarations
real(8),allocatable,dimension(:,:,:),codimension[:]::fIn,fOut,fEq
! Find total image numbers and use it for domain decomposition
img=num_images()
! Global Allocations
allocate(fIn(0:8,lx,ly)[*])
allocate(fOut(0:8,lx,ly)[*])
allocate(fEq(0:8,lx,ly)[*])
! Program subroutines
sync all
! Propagate subroutine
im=this_image()
do n=1,ly
    fIn(1,1,n)=fOut(1,lx,n)[im-1]
    fIn(3,lx,n)=fOut(3,1,n)[im+1]
    fIn(5,1,n)=fOut(5,lx,n-1)[im-1]
    fIn(6,lx,n)=fOut(6,1,n-1)[im+1]
    fIn(7,lx,n)=fOut(7,1,n+1)[im+1]
    fIn(8,1,n)=fOut(8,lx,n+1)[im-1]
enddo
```

The LBM parallelization is based on domain decomposition (DD) technique. We divided whole computational domain to subdomains. To take the advantage of the all the cores available to us, the number of the subdomains and cores are equated. These subdomains consist of columns of grids called strip as in Figure 2**.** Therefore, in the propagation (streaming) step, only the neighbor strips send and fetch data from each other.

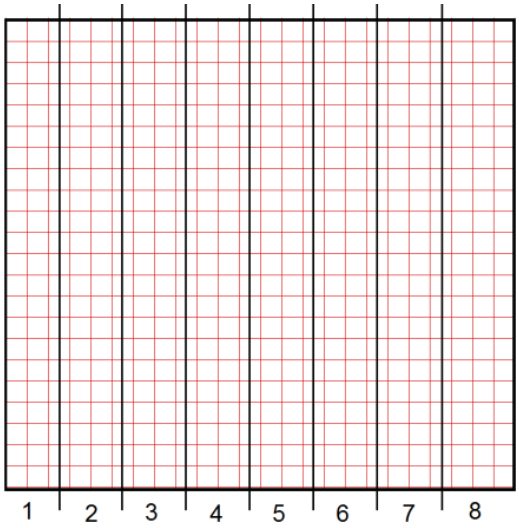


Figure 2 Grid partitioning for 8 images

4 Test problems and results

The lid-driven square cavity and the channel flows are important benchmark problems to validate newly developed or modified methods in computational fluid dynamics (CFD) [7]**.** Intel Fortran Compiler XE 2011 with CAF support is used to develop our both sequential and parallel solvers. Operating system is Windows XP Professional SP3 64 Bit. The computational platform has a total main memory of 4 GB. The same test problems are run 1000 times in both sequential and parallel LBM solvers. Using domain decomposition method, the computational domain is divided to 2, 4, and 8 strips in order to reach maximum performance over the multi core computational platform.

4.1 Lid-driven square cavity flow

In a cavity flow, a viscous fluid is confined by a square box. Top boundary keeps moving in one direction mostly with a constant speed. Other walls do not move. Because of the shear stress, momentum and energy are transferred between fluid and walls. Consequently, the fluid inside the cavity starts moving. After a while flow becomes steady. Although these flows are rather simple in geometry, they display all the details of the fluid flows.
In the validation phase, we simulated a lid-driven square cavity flow in 2-dimensions using LBM. The dimension of the square lattice and the speed of the lid are 128^2 and 0.05 in LBM units respectively. The Reynolds number of the test problem must be equal to 100. Kinematic viscosity is obtained as,

$$\gamma_{LBM} = \frac{U_{LBM} \times N}{\mathrm{Re}} = 0.064$$

in LBM units. Here, *N* depicts the number of lattice points. The single relaxation time (τ) for the D2Q9 model calculated by the equation stated below,

$$\tau = 3\,\gamma_{LBM} + 0.5 = 0.69$$

In the study of cavity flow, each processor is assigned to 128/n columns of grids which n is the number of cores. The solutions from both sequential and parallel LBM solvers match exactly as shown in Figure 3.

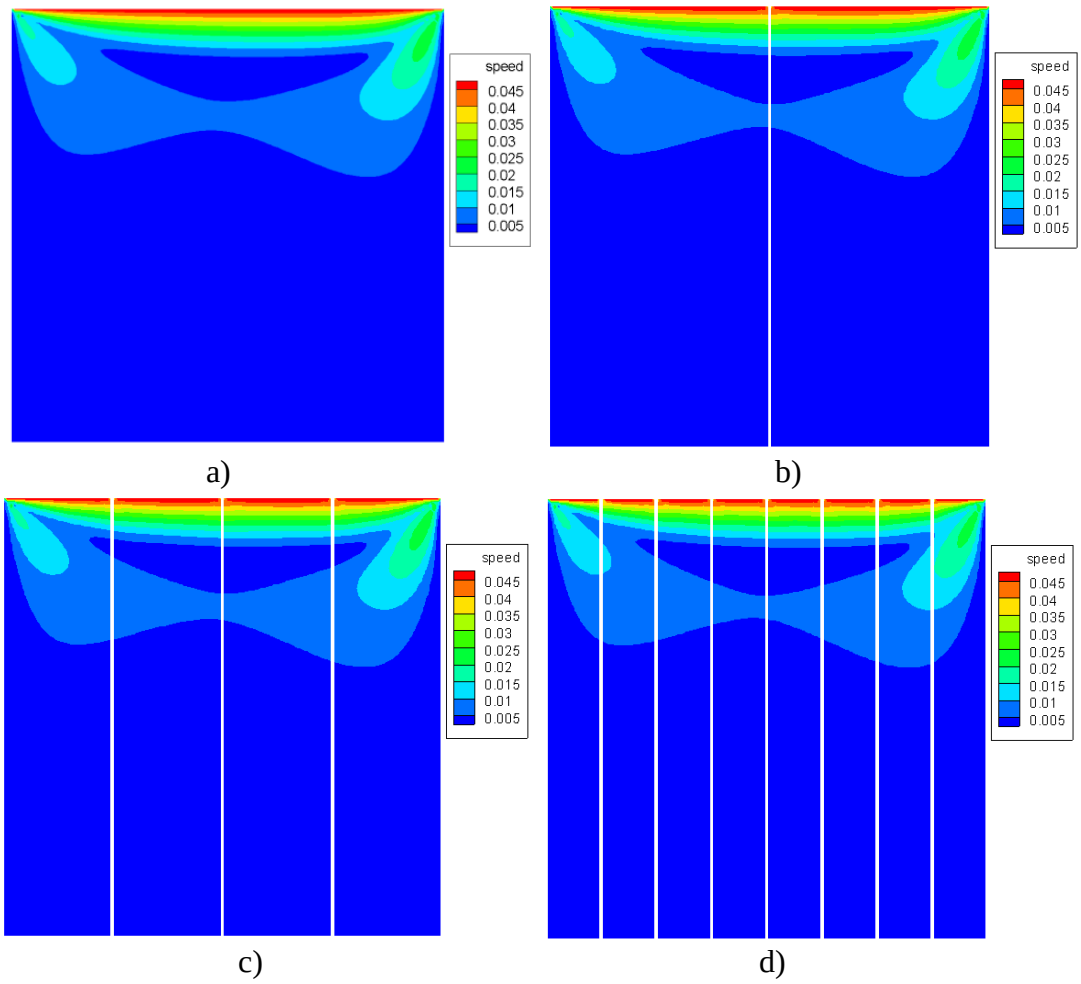


Figure 3 Result of speed contours (a) Sequential solver (b) 2 core parallel solver (c) 4 core parallel solver (d) 8 core parallel solver

4.1.1 Boundary Conditions

Bounce back is used to model solid stationary or moving boundary condition, nonslip condition, or flow-over obstacles. In this study bounce back boundary condition is implemented to LBM solver for stationary wall boundaries. The method mainly implies that an incoming particle towards the solid boundary bounces back into flow domain,

$$f_5 = f_7 \quad f_2 = f_4 \quad f_6 = f_8 \quad f_1 = f_3$$

It is important to note that in our application the boundary condition is applied before the streaming process. The bounce back ensures conservation of mass and momentum at the boundary. During the test problem is performed, the known velocity component is used to define moving lid boundary. Zou and He [22] described a method to calculate three unknown distribution functions based on

$$\rho = \sum_{k=0}^{8} f_k$$

and

$$\rho u = \sum_{k=0}^{8} f_k c_k$$

with equilibrium conditions assumption, normal to the boundary. Here (c_k) shows the lattice (microscopic) velocities,

$$\rho = \frac{1}{1+u_y}[f_0 + f_1 + f_3 + 2(f_2 + f_6 + f_5)]$$

$$f_4 = f_2 - \frac{2}{3}\rho u_y$$

$$f_7 = f_5 + \frac{1}{2}(f_1 - f_3) - \frac{1}{6}\rho u_y - \frac{1}{2}\rho u_x$$

$$f_8 = f_6 + \frac{1}{2}(f_3 - f_1) - \frac{1}{6}\rho u_y + \frac{1}{2}\rho u_x$$

After validation of the parallel solver with sequential solver, we continued our tests with performance studies of the parallel solver. We realized three more tests using 256^2, 384^2 and 448^2 lattice points. In each test we used 2, 4 and 8 cores for our parallel LBM solver. The solution times and parallel efficiency (%) values of the parallel LBM solver for 1000 iterations are shown in Figure 4.Parallel efficiencies change between 63 % and 109 %. The lowest efficiency figure (63 %) comes from the test consists of the least lattice points (128x128) and the highest sub domain (8) number. It can be explained with the high ratio of the core-to-core communicating lattice columns (=7) and the low number of internal lattice columns (=16). We obtained an efficiency of 91 %, when we used 448x448 lattice points and the maximum core number of our workstation which is 8. In this case, the number of core-to-core communicating lattice columns was same, while internal lattice columns were increased to 56. The lower ratio of the communication overhead is resulted with a higher parallel efficiency Figure 4. The speed-up value for this case is calculated as 7.28 (=525/72.1).

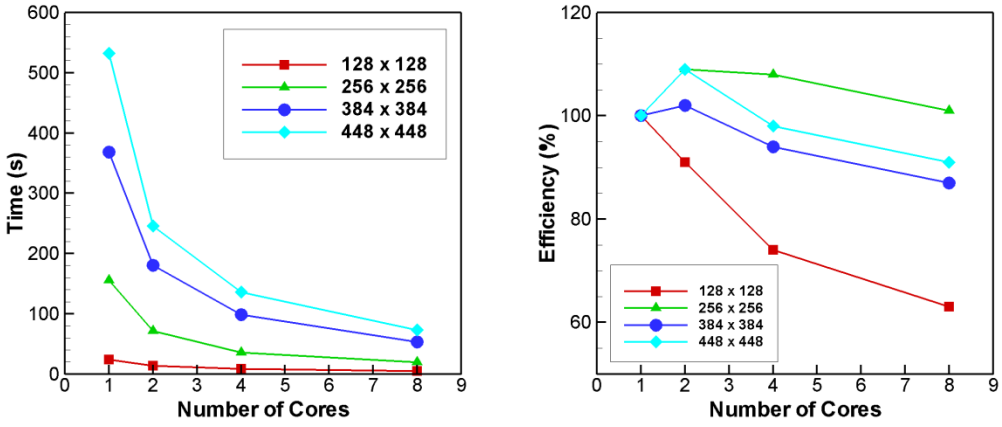


Figure 4 Multi-core parallel LBM solver performance values for cavity flow (a) wall time (b) parallel efficiency

4.2 Channel flow

In a channel flow, a viscous fluid flows along a thin tube with a pressure difference. In the study top and bottom boundaries are kept stationary and the west and east boundaries are kept opened. The pressure BC are applied at input and output. Like the lid driven cavity, the channel flow is rather simple in geometry, however, it displays all the details of the fluid flows.

In the validation phase, we simulated a channel flow in 2D using LBM. The dimension of the channel lattice and the maximum inlet speed are 512×50 and 0.1 in LBM units respectively. The Reynolds number of the test problem must be equal to 100. Kinematic viscosity is calculated by,

$$\gamma_{LBM} = \frac{U_{LBM} \times N}{\mathrm{Re}} = 0.05$$

in LBM units. The characteristic relaxation time (τ) for the D2Q9 model is calculated using the equation stated below,

$$\tau = 3\,\gamma_{LBM} + 0.5 = 0.65$$

again in LBM units.

Like the lid- driven cavity problem, the channel flow problem was also iterated with 1000 time steps. The computational domain is also divided along x-direction and the division is considered at 2, 4, and 8 strips. Each processor has given 512/n columns of grids which n is the number of cores. The solutions from both sequential and parallel LBM solvers match exactly as shown in Figure 5**.**

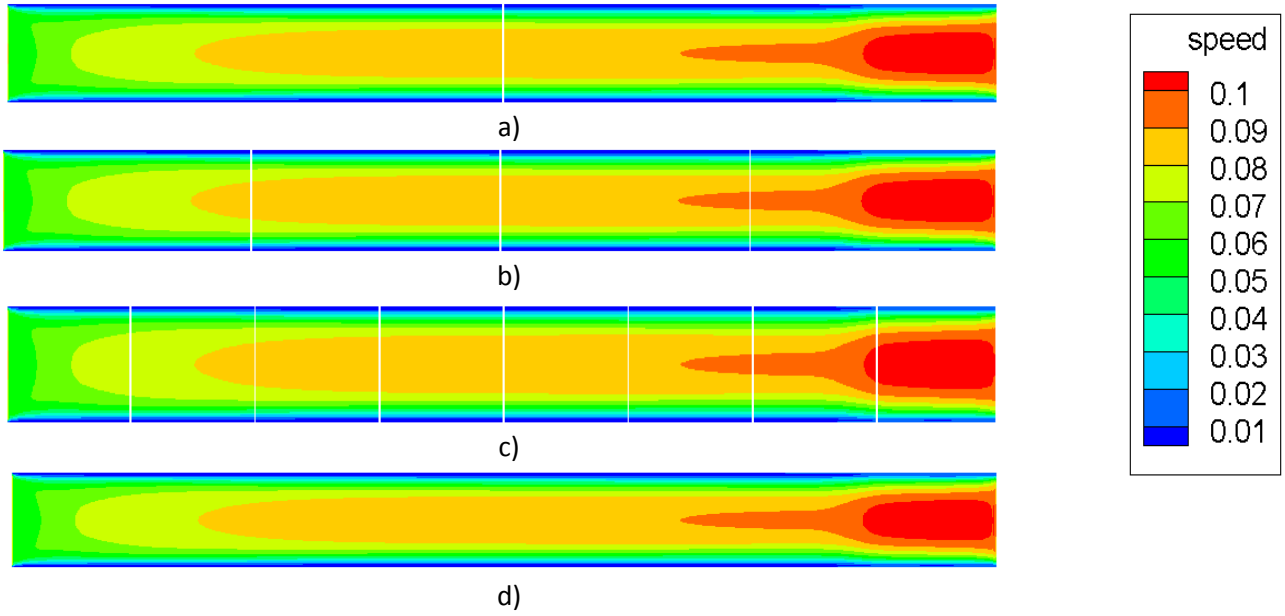


Figure 5 Result of speed contours. a) 2 core parallel, b) 4 core parallel, c) 8 core parallel, d) Sequential solver

4.2.1 Boundary Conditions

Bounce back boundary conditions are also used to model solid stationary walls of the channel. The details of the boundary condition is given previous section. For the channel problem pressure (Dirichlet) boundary condition, defined by Zou and He [22] is successfully implemented,

$$\rho_0 = \sum_{k=0}^{8} f_k$$

and

$$u = \frac{1}{\rho_0} \sum_{k=0}^{8} f_k c_k$$

Density (ρ_0) is calculated the velocity distribution functions. Using this density, we calculated the velocity components. At the wall boundary nodes the velocity tangent of the particle is zero. The component of velocity normal to the boundary is solved. In addition to the macroscopic velocity, a proper distribution function requires to determine at the boundary nodes. After the streaming step, there are three unknown directional densities at each lattice node pointing from the boundary into the domain. The missing distribution functions can be specified in a way that maintains the specified density ρ_0 at their lattice nodes.

$$u_x = -1 + \frac{1}{\rho_{outlet/inlet}} [f_0 + f_1 + f_3 + 2(f_2 + f_6 + f_5)]$$

$$f_3 = f_1 - \frac{2}{3} \rho_{outlet/inlet} u_x$$

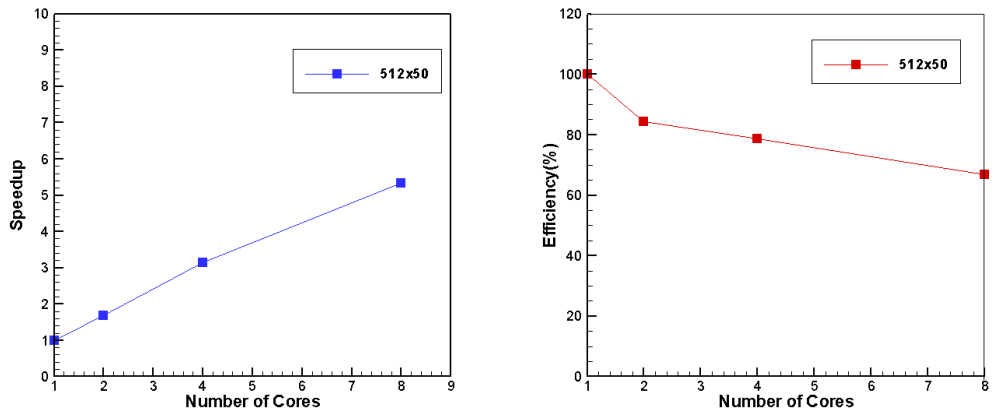
$$f_7 = f_5 + \frac{1}{2}(f_2 - f_4) - \frac{1}{6} \rho_{outlet/inlet} u_x$$

$$f_6 = f_8 - \frac{1}{2}(f_2 - f_4) - \frac{1}{6}\rho_{outlet/inlet}u_x$$

After validation of the parallel solver with sequential solver, we continued our tests with 512×50 lattice points for performance study of the parallel solver. In each test we used 2, 4 and 8 cores for our parallel LBM solver. The solution times and parallel efficiency (%) values of the parallel LBM solver for 1000 iterations are shown Figure 6. The maximum parallel efficiency is obtained as 84% for two cores. The speed-up values are represented in Table 1 and efficiency and speed up plots are shown in Figure 6,

Table 1 Channel flow parallelization features

Number of cores	CPU(sec)	Speedup	Efficiency
1	33,04	1	1
2	19,60	1,68	0,84
4	10,48	3,15	0,78
8	6,18	5,34	0,66



a) b)

Figure 6 Multi-core parallel LBM solver performance values for channel flow (a) speedup (b) parallel efficiency.

4 Conclusion and Future Work

In this study as a first step, a LBM solver based on D2Q9 model is parallelized using Coarray Fortran. Coarray Fortran is adapted due to its simple syntax and short development time in addition to the easy debugging. Consequently, the developed sequential LBM solver is modified with much less effort than the conventional parallelization techniques are required. In the second phase of our work, our parallel solver is validated with data coming from the sequential solver. Next, the solution times of the parallel and sequential solvers are recorded for performance comparisons. The study is focused on the small scale core numbers and limited with a maximum of 8 cores. For a 448^2 lattice points case, the parallel cavity LBM solver is approximately 7.28 times faster than sequential one workstation with 8 cores. The parallel efficiency of the parallel LBM solver for the same test case is 91%. As a second test problem the channel flow is chosen. Same procedure is pursued for the implementation of LBM solver to the channel flow. The contradictory relation can be seen easily from the efficiency and speedup plots where the speedup increases with number of cores, the efficiency diminishes. Although

still in the development phase, as a consequence Coarray Fortran is an important alternative over the conventional parallelization tools.

References

[1] D. Bespalko, A. Pollard, M. Uddin ,D.J.K. Mewhort, N.M. Cann, G.W. Slater, et al. (Eds.) *High Performance Computing Systems and Applications*, Lecture Notes in Computer Science, 5976, 1-19. 2010.

[2] B.D. Chirila, Introduction to Lattice Boltzmann Methods, University of Wuerzburg. 2010.

[3] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, Experiences with Sweep 3D implementation in Co-array Fortran, *J. Supercomputer, 36,*101-121. doi: 10.1007/s11227-006-7952-7. May, 2006.

[4] Y. Dotsenko, C. Coarfa, J. Mellor-Crummey, D. Chavarria-Miranda. Experiences with Co-Array Fortran on Hardware Shared Memory Platforms. *In Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2004)*, West Lafayette, Indiana, September 2004.

[5] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-Gas Automata for the Navier-Stokes Equation, *Phys. Rev. Lett.,56*, 1505. doi:http://dx.doi.org/10.1103/PhysRevLett.56.1505. April 7, 1986.

[6] M. Hasert, H. Klimach, S.P. Roller, CAF versus MPI - Applicability of Coarray Fortran to a Flow Solver: Recent Advances in the Message Passing Interface, *Lecture Notes in Computer Science, 6960*, 228-236. 2011.

[7] S. Hou, Q. Zou, S. Chen, G. Doolen, A.C. Cogley, Simulation of Cavity Flow by the Lattice Boltzmann Method, *J. Comp. Physics, 118 (2),*329–347. May, 1995.

[8] D. Kandhaia, A. Koponenb, A.G. Hoekstraa, M. Katajab, J. Timonenb, P.M.A. Sloota, Lattice-Boltzmann hydrodynamics on parallel systems, *Computer Physics Communications ,111 (1–3),* 14–26. June, 1998.

[9] G.M. Karniadakis and A. Beskok *Micro Flows Fundamentals and Simulation*. Verlag, NY: Springer. 2002.

[10] Q. Li, C. Zhong, K. Li, G. Zhang, X. Lu, Q. Zhang, K. Zhao, X. Chu, A parallel lattice Boltzmann method for large eddy simulation on multiple GPUs, *Computing, 96 (6)*, 479-501. June, 2014.

[11] G.R. McNamara, and G. Zanetti, Use of the Boltzmann Equation to Simulate Lattice-Gas Automata, *Phys. Rev. Lett. 61,* 2332. November 14, 1988.

[12] R.W. Numrich, J. Reid, Co-array Fortran for parallel programming, *ACM SIGPLAN Fortran Forum 17 (2),*1-31. 1998. doi: 10.1145/289918.289920.

[13] C. Obrecht, F. Kuznik, B. Tourancheau, and J.J. Roux, Multi-GPU implementation of the lattice Boltzmann method, *Computers and Mathematics with Applications, 10,*1016. 2011.

[14] G. Punzo, F. Massaioli, and S. Succi, High resolution lattice–Boltzmann computing on the IBM SP1 scalable parallel Computer, *Journal Computers in Physics, 8 (6),* 705-711. November/December, 1994. doi: 10.1063/1.168487.

[15] D. Raabe, Overview of the lattice Boltzmann method for nano- and micro scale fluid Dynamics in materials science and engineering, *Modeling Simul. Mater. Sci. Eng., 12,*13-46. 2004.

[16] K. Sembritzki, Evaluation of the Coarray Fortran Programming Model on the Example of a Lattice Boltzmann Code, Masters Thesis in Computational Engineering, University of Erlangen, Nürnberg. 2012.

[17] S. Succi, *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation)* (Reprint edition). Oxford University Press. July 18, 2013.

[18] P.M. Tekic, J.B. Radjenovic, M. Rackovic, Implementation of the Lattice Boltzmann Method on Heterogeneous Hardware and Platforms using OpenCL, *Advances in Electrical and Computer Engineering, 12 (1),* 51-56. 2012.

[19] J. Tölke, Implementation of a lattice Boltzmann kernel using the Compute Unified Device

Architecture developed by NVIDIA, *Computing and Visualization in Science, 13 (1),*29-39. January, 2010.

[20] D. Vidal, R. Roy, F. Bertrand, A parallel work load balanced and memory efficient lattice-Boltzmann algorithm with single unit BGK relaxation time for laminar Newtonian flows, *Computers &Fluids, 39,*1411-1423. 2010.

[21] G. Zhao-Li, Z. Chu-Guang, S. Bao-Chang, Non-equilibrium extrapolation method for velocity and pressure boundary conditions in the Lattice Boltzmann method, *Chinese Physics, 11 (4),* 366-374. 2002.

[22] Q. Zou and X. He. On pressure and velocity boundary conditions for the lattice Boltzmann *BGK model. Phys. Fluids 9 (6), 1591-1598*. June 1997.