**Proceedings of the ASME 2009 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC/CIE 2009
August 30-September 2, 2009, San Diego, USA**

DETC2009/MESA-12345

PARALLELISATION OF CLASSICAL LBM USING OPENACC AND EVALUATION ON 2D LID-DRIVEN CAVITY TEST

Kartik Ruikar
Department of Ocean Engineering
Indian Institute of Technology Madras
Chennai, India - 600036
Email: na20b053@smail.iitm.ac.in

ABSTRACT

This report outlines a parallelization approach leveraging OpenACC to enhance the performance of a classical Lattice Boltzmann Method (LBM) solver. The methodology is validated through simulations of a 2D lid-driven cavity test case. By harnessing OpenACC's capabilities, significant computational speed-up is achieved, demonstrating the effectiveness of the proposed parallelization technique. The study not only presents the implementation details of parallelization but also evaluates its performance against a standard benchmark problem.

NOMENCLATURE

f velocity distribution function at a given lattice point.
ζ particle velocity.
τ relaxation time.
f^{eq} velocity distribution function in equilibrium state.
ω_α weight for the respective resolved velocity component.
C_α Lattice speed for respective direction.
ρ fluid density.
C_s speed of sound.
$\vec{u}$ macroscopic velocity vector.
ρ_N density at the north wall boundary.
$\vec{u}_N$ horizontal velocity at the north wall boundary.
$\vec{v}_N$ vertical velocity at the north wall boundary.

INTRODUCTION

This project report presents the implementation of a classical Lattice Boltzmann Method (LBM) in C++ and its parallelization using OpenACC. The LBM solver exhibits significant potential for parallelization, particularly in the collision and streaming steps, which collectively consume around 90% of the total simulation time. To facilitate efficient computation, an abstract data structure is employed to store distribution values at each grid point. The implementation begins with the initialization of velocity, density, and distribution function vectors, followed by the execution of collision, streaming, and boundary condition application steps. The process iterates until a convergence criterion based on the error in velocities within a specified tolerance is met. Two main blocks of parallelism are utilized: the first block handles memory allocation for all variables and initialization on the GPU, while the second block operates within a while loop, managing iterations until convergence. It performs streaming and collision step updates on GPU and transfers error values. Upon convergence, the GPU transfers the complete solution to the CPU for further post-processing.

Following the introduction, subsequent section of the paper delve into aspects of the classical LBM solver. Section 3 provides an in-depth discussion on the parallelization methodology employed, elucidating the strategies implemented to leverage OpenACC effectively. In Section 4, the numerical setup used for the 2D lid-driven cavity test is described, encompassing parameters, boundary conditions, and computational domain specifications. Subsequent sections focus on presenting the achieved speed-up

results attained through the parallelized solver and its comparison with benchmark Direct Numerical Simulation (DNS) outcomes. Through systematic analysis, this project aims to evaluate the efficacy of the proposed parallelization approach in improving the computational efficiency of LBM simulations.

CLASSICAL LBM

In the classical Lattice Boltzmann Method (LBM), fluid dynamics are simulated by representing fluid particles as entities moving across a lattice grid. The method involves iteratively solving the lattice Boltzmann equation, which is derived from the Boltzmann equation, for the particle velocity distribution function. Various models are utilized to represent the velocity distributions, such as D2Q9 and D2Q4, where 'D' denotes the number of dimensions and 'Q' represents the number of velocity components. These models determine the spatial discretization and the number of discrete velocity directions considered in the simulation. The following lattice boltzmann equation using BGK collision model is modeled in this project

$$\frac{\partial f}{\partial t} + \zeta.\nabla f = -\frac{1}{\tau}[f - f^{eq}] \tag{1}$$

The simulation process consists of two main steps: collision and propagation. During collision, particles interact and adjust their velocities based on simplified collision rules, while in propagation, particles move to neighboring lattice nodes according to their updated velocities. By repetitively executing these collision and propagation steps, the LBM effectively captures the macroscopic behavior of fluid flow.

Collision step

In the collision step of the lattice Boltzmann method (LBM), particle distribution functions are relaxed towards local equilibrium, incorporating collision effects.

$$f_\alpha^* = \omega * f_\alpha^{eq} + (1-\omega) * f_\alpha \tag{2}$$

where, f_α^{eq} is calculated as follows

$$f_\alpha^{eq} = \omega_\alpha \rho[1 + \frac{C_\alpha.\vec{u}}{C_s^2} + \frac{(C_\alpha.\vec{u})^2}{2C_s^4} - \frac{\vec{u}.\vec{u}}{2C_s^2}] \tag{3}$$

Streaming step

In the streaming step, these intermediate distributions propagate to neighboring lattice nodes, simulating the movement of particles according to their respective velocities.

$$f_\alpha = f_\alpha^* \tag{4}$$

PARALLELISATION METHODOLOGY

Efficient parallelization is crucial for accelerating model convergence. Here, parallelization methodology implemented in C++, leveraging OpenACC directives for seamless integration of parallel computing techniques is presented. The approach focuses on two main blocks: initialization and the main iteration loop. Within these blocks, the parallel loops are employed to distribute computation across available processing units effectively. Additionally, function segregation enhances code modularity and facilitates parallel execution of critical tasks.

1. Initialisation:
   (a) Velocity, density, and distribution function vectors and all the constants required for the simulation are defined.
   (b) Initial data is copied to system memory using the copy data clause, ensuring its copied out after the solution is converged. The scope is throughout the simulation convergence.
   (c) Constants and coefficient vectors are only copied in using the copyin clause.
2. Initial Parallel block :
   (a) Parallel for loops are implemented using # pragma acc parallel loop.
   (b) The collapse argument is utilized for paralleling closely nested loops.
3. Parallelization within Main Loop:
   (a) Within the main while loop (iterating until velocity error converges below a tolerance), at each iteration the velocity error and iterations value stored in variable is copied on to the system and copied out at the end of the iteration to compare with tolerance on CPU.
   (b) Parallel loops are employed for collision, streaming, boundary condition application, and error calculation steps using the parallel loop clause.
   (c) Data dependencies are not present in loops, allowing for efficient parallel execution.
4. Function Segregation:
   (a) Functions for equilibrium distribution calculation and boundary condition application are separated.
   (b) Parallelization within these functions is achieved using parallel for loop clause, avoiding redundant copyin of data by using present clause for input vectors.

Careful management of data movement between device and host memory ensures efficient computation. Parallel loops are optimised for the number of gangs, workers by trying out different combinations. The optimal time was found at default setting of 1 gang and 128 vectors.

NUMERICAL SETUP

A 2D square domain of 0.2 x 0.2 m is modeled. D2Q9 model is used for modelling the particle velocity and Maxwells distribution function to model their distributions. The lid velocity is kept 0.1 and density is initialised to 5.0. The flow of reynolds number (Re) of 1000 is simulated. The domain is divided into 100 lattice points in both directions (nx, ny denoting lattice points in x and y direction). The convergence rate was tested till 500 lattice points.
The boundary conditions of bounce back is used for fixed bottom and side walls. Whereas for top moving wall, the moving wall boundary condition is used.

1. South wall

$$f_3 = f_1, f_6 = f_8, f_7 = f_5. \tag{5}$$

2. East wall

$$f_4 = f_2, f_8 = f_6, f_7 = f_5. \tag{6}$$

3. West wall

$$f_2 = f_4, f_6 = f_8, f_5 = f_7. \tag{7}$$

4. North wall

$$\rho_N = \frac{1}{1+v_N}[f_0 + f_2 + f_4 + 2*(f_3 + f_7 + f_6)] \tag{8}$$

$$f_1 = f_3 \tag{9}$$

$$f_8 = f_6 + \frac{f_2 - f_4}{2} - \frac{\rho_N u_N}{2} \tag{10}$$

$$f_5 = f_7 + \frac{f_4 - f_2}{2} + \frac{\rho_N u_N}{2} \tag{11}$$

Tolerance value of 1e-8 is used for convergence and the velocity error is computed by summing the square root of the square of the deviation from the mean value for all lattice points, which is then normalized by the total number of lattice points. This measure provides insight into the stability and accuracy of the simulation, helping to ensure that the computed velocities converge to a consistent solution within the specified tolerance level.

RESULTS

Initially, a serial implementation of the model is developed, and its results are validated against benchmark Direct Numerical Simulation (DNS) results [1], utilizing a Reynolds number of 1000. The convergence behavior for a lattice grid size of 100x100 is illustrated in Figure 1. Validation of the results is performed by computing the maximum and minimum vertical velocities within the domain, normalized by the inlet velocity -

$$v_{max} = 0.3769447 \tag{12}$$

$$v_{min} = -0.5270773 \tag{13}$$

Figure 2 and Figure 3 depict the velocity profiles for horizontal and vertical velocities plotted using the software ParaView, respectively, which were compared against the results obtained from Direct Numerical Simulation (DNS). In order to verify the consistency of results between the parallel and serial implementations, the velocities at x = 0.1 and y = 0.1 were plotted and compared. These comparisons are illustrated in Figure 4 and Figure 5. It was observed that the results obtained from both the parallel and serial codes matched perfectly, ensuring the correctness and reliability of the parallel implementation.
The speedup achieved was calculated to be 22.54 when utilizing the optimal configuration of gangs, workers, and vectors. In this configuration, the parallel program converged in 7.59 seconds, whereas the equivalent serial program required 171.08 seconds to reach the same results. This significant speedup demonstrates the effectiveness of parallelization in improving computational efficiency. Furthermore, the convergence time of the simulation was analyzed across varying numbers of lattice points, as illustrated in Figure 6

CITING REFERENCES

[1] 2-D Lid-Driven Cavity Flow Benchmark Solutions at https://www.acenumerics.com.

ACKNOWLEDGMENT

Thanks to Dr. Kameshwar Rao Anupindi for his guidance in the course on parallel computing, as well as to the Teach-

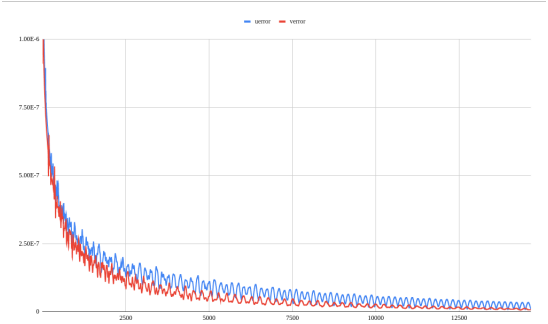

FIGURE 1. Convergence

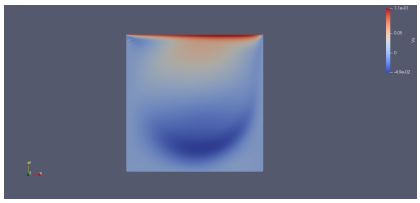

FIGURE 2. Horizontal velocity contours

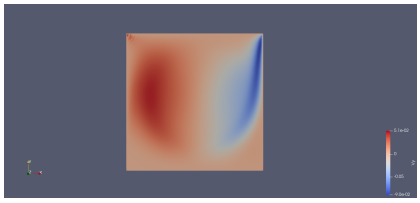

FIGURE 3. Vertical velocity contours

ing Assistants for their invaluable assistance in resolving doubts throughout the duration of the course.

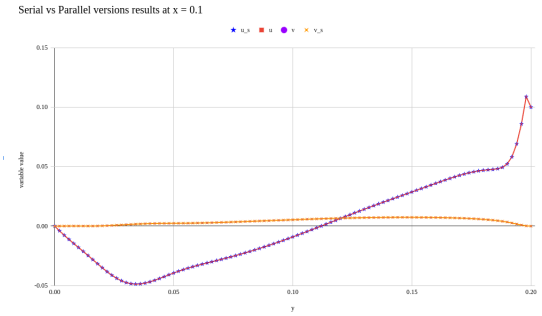

FIGURE 4. Velocity results at x=0.1

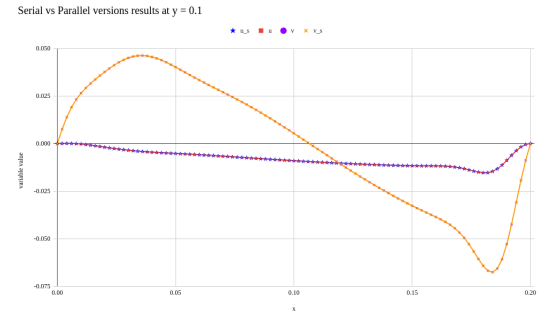

FIGURE 5. Velocity results at y=0.1

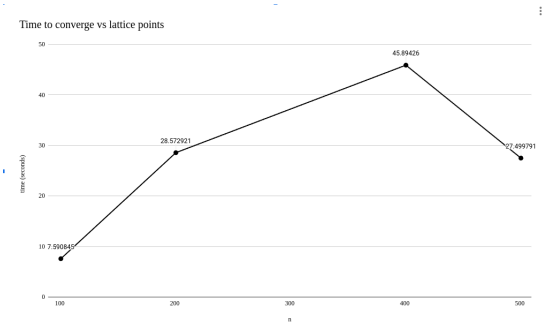

FIGURE 6. Convergence time against no. of lattice points