ISSN 2176-5480

PERFORMANCE ANALYSIS OF SERIAL AND PARALLEL IMPLEMENTATION FOR A 2D MICROFLUIDICS SIMULATOR USING THE LATTICE BOLTZMANN METHOD

Fabíola Martins Campos de Oliveira
Luiz Otávio Saraiva Ferreira
Department of Computational Mechanics, Mechanical Engineering Faculty, State University of Campinas, Campinas, Brazil
fabiola@fem.unicamp.br, lotavio@fem.unicamp.br

Abstract. *Recent advancements on multicore processors technology led to parallel computing, decreasing runtime of several interesting problems for engineering, and making possible an expansion of the problem domain. One example is the fluid mechanics, an area of great economic and academic interest, whose simulations have high computational cost. This said, a good method for simulating fluid flows with proven advantage for use in parallel computing is the Lattice Boltzmann Method. As its algorithm is highly parallelizable, simulations based on this method tend to gain efficiency when more cores are used. As a first step, this work gets a cpu-based algorithm for measuring efficiency running on a single-core processor and, later, this algorithm is converted into a parallelized cpu-based code running on a multicore processor. At last, the performance of both fluidic simulators based on the Lattice Boltzmann Method running on a single-core processor and on a multicore processor is compared.*

Keywords: *Lattice Boltzmann method, microfluidics, parallel computing*

1. INTRODUCTION

From 1986 until 2002, software developers and final users could rely on technical advances of microprocessors to increase performance of their programs, having an average of 50% more speed per year (Pacheco, 2011). After 2002, this performance decreased to 20% per year due to technical issues, like the difficulty of dissipating heat of high-density transistors microprocessors. To overcome this, the next step for industries to achieve higher performance was adding more than one processor on a single chip. The main consequence of this decision for programmers was that their old programs would no longer benefit from new technologies as before, since single processor programs do not recognise multiple processors. Since 2005, when most companies started offering multicore processors, serial codes needed to be rewritten in order to increase performance using these new processors. There is a trend for developing translation programs, which would automatically convert a serial code into a parallel code, but since this approach has only been good for specific cases, it is needed to find more efficient algorithms for each case (Pacheco, 2011).

For mechanical engineering, an area that widely takes advantage of parallelization is fluid mechanics. Usually, traditional computational fluid dynamics (CFD) methods demand heavy computational resources in order to simulate fluid flows properly. A more powerful method for solving fluid dynamics problems (Mohamad, 2011) that is effectively parallelizable is the Lattice Boltzmann method (LBM). Compared to finite differences method, LBM have proved to double performance (Chen *et al.*, 1994) LBM models the fluid as particles probabilities distribution functions, that collide and propagate over a lattice domain (Oliveira and Ferreira, 2012). This method easily handles features that traditional CFDs can not deal with or is very slow, like complex boundaries and multicomponent multiphase flows (Succi, 2001).

This work measures performance of a basic solver using the Lattice Boltzmann method, with solid walls, comparing a pure serial code and a parallel version using Message-Passing Interface (MPI), an extension to languages like C and C++ suitable for distributed-memory systems, meaning that each core has its own amount of memory, and therefore communication among cores is required. As results showed that the parallel code is advantageous, there is expectation on reusability of code for future implementations.

2. MESSAGE-PASSING INTERFACE (MPI) PROGRAMMING

MPI is an explicit parallel extension, what means that the work of each core must be specified, being a more powerful tool than higher level languages, like OpenMP (Pacheco, 2011). The MPI extension includes type definitions, functions and macros and is well suitable for distributed-memory systems, offering ways for communication among cores. There are mainly two ways to parallelize a program: using task-parallelism or data-parallelism (Pacheco, 2011). Task-parallelism divides the problem in several tasks that are distributed among cores, while data-parallelism divides the problem data among cores, and they execute similar tasks on its own data. Writing parallel programs involves coordination of cores: they usually need to communicate with each other, sending information or data. It is also desired for the program to have load balancing, meaning that all cores should receive aproximately the same amount of work, so there are not idle cores during execution of the program. Another type of coordination is syncronization: cores sometimes need to wait for all the other cores to reach the end of a point in code before proceed.

Data can be divided in three ways: using a block partition, when first data block *data/processes* is assigned to first pro-

ISSN 2176-5480

cess, second data block *data/processes*, to second process, and so on; using a cyclic partition, like round-robin scheduling or a block-cyclic partitioning, when blocks of data are assigned to process in a round-robin manner. Most basic MPI functions works with block partition, but, to have cyclic or block-cyclic partitioning, a MPI derived data type can be created in order to communicate different partitioning.

Lastly, the terms concurrent, parallel and distributed computing have slight differences. While concurrent computing is most used for a program whose multiple tasks can be run at any time, parallel computing means that multiple tasks of a program cooperate to solve a problem. Distributed computing is used for a program that needs to help other programs to solve a problem.

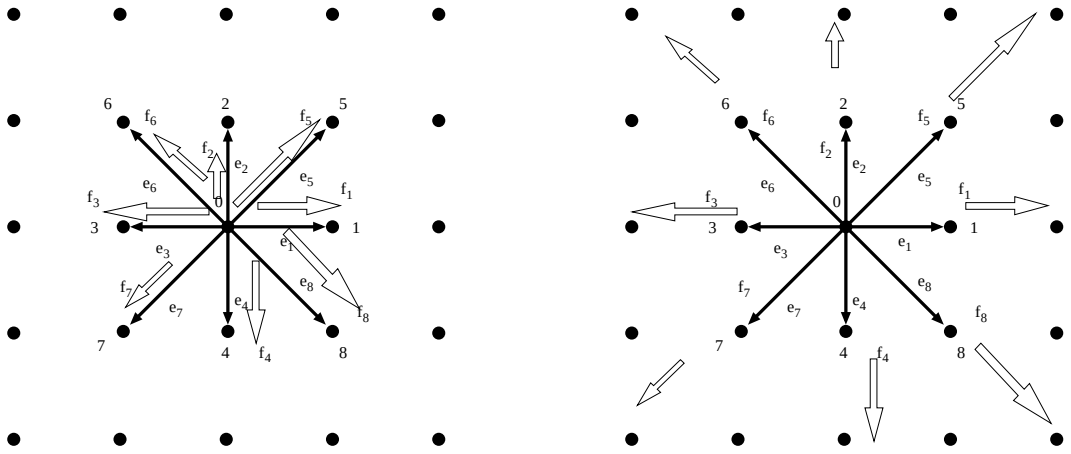
3. NUMERICAL MODELS AND THE LATTICE BOLTZMANN METHOD

The Lattice Boltzmann method (LBM) derived from the Lattice Gas Cellular Automata (LGCA) method of fluid-flow simulation (Wolf-Gladrow, 2005). LBM recovers Navier-Stokes equations in the macroscopic scale based on Boltzmann kinetic theory (Succi, 2001).

In both LGCA and LBM methods, simulation is separated in two steps: streaming and collision. Discretizing the original Boltzmann equation on time, space and momentum gives the LBM Eq. (1) (Aidun and Clausen, 2010; Zhang, 2011):

$$f_a(\mathbf{x} + e_a\Delta t, t + \Delta t) = f_a(x,t) - \frac{[f_a(x,t) - f_a^{eq}(x,t)]}{\tau} \tag{1}$$

in which x is the position of the particle, e_a is its microscopic velocity, t is the time, Δt is the time-step of the simulation, f_a is the particle probability distribution function on direction a, $f_a(\mathbf{x} + e_a\Delta t, t + \Delta t) = f_a(x,t)$ is the streaming part, $f_a^e q$ is the equilibrium probability function, τ is the relaxation parameter and $\frac{[f_a(x,t) - f_a^{eq}(x,t)]}{\tau}$ is the collision term, which is the simplified model introduced in 1954 by Bhatnagar, Gross and Krook and known as BGK aproximation. One of the most used LBE models is the D2Q9 (2 dimensions and 9 velocities), in which the microscopic velocity e_a $(a = 0, \ldots, 8)$ is restricted to 8 directions plus a rest particle, 3 magnitudes, and there is a single particle mass, as shown in Fig. 1a (Oliveira and Ferreira, 2012). It was assumed on Eq. (1) that particle mass $= 1$, so that microscopic velocities and momenta are equivalent.



(a) D2Q9 lattice and microscopic velocities

(b) Streaming step of D2Q9 lattice

Figure 1: LBM model and streaming step. Each velocity e_a (black arrows) has an associated frequency f_a (white arrows)

Units of length and time measurement are the lattice unit (lu) and the lattice time (lt), respectively. Velocity magnitudes of e_1 through e_4 is $1lu/lt$, and velocity magnitudes of e_5 through e_8 is $\sqrt{2}lu/lt$.

The sum of distribution functions of a lattice node gives the macroscopic fluid density:

$$\rho = \sum_{a=0}^{8} f_a \tag{2}$$

and the macroscopic velocity is computed as Eq. (3):

$$u = \frac{1}{\rho}\sum_{a=0}^{8} f_a e_a \tag{3}$$

On the streaming step, each function f_a goes to the nearest neighbor lattice node pointed by the corresponding arrow, as shown in Fig. 1b, resulting on updated values of f for each node. Next step is the collision, and first it is need to perform calculation of the equilibrium distribution function f^{eq} for each node of the lattice, using Eq. (4):

$$f_a^{eq}(x) = \omega_a \rho(x) \left[1 + 3\frac{e_a u}{c^2} + \frac{9}{2}\frac{(e_a u)^2}{c^4} - \frac{3}{2}\frac{u^2}{c^2} \right] \tag{4}$$

in which ω_a is the weight for each particle: $4/9$ for $a = 0, 1/9$ for $a = 1, 2, 3, 4$ and $1/36$ for $a = 5, 6, 7, 8$ and c is the lattice sound speed, usually $1lu/lt$. After that, BGK aproximation is calculated using this result. Besides the two steps of the method, boundary conditions should be applied on each iteration, so that it is possible to consider the effect of solid walls in the flow. The most common boundary condition for walls is the bounceback condition, as shown in Fig. 2. Periodic condition can be applied on domain edges.

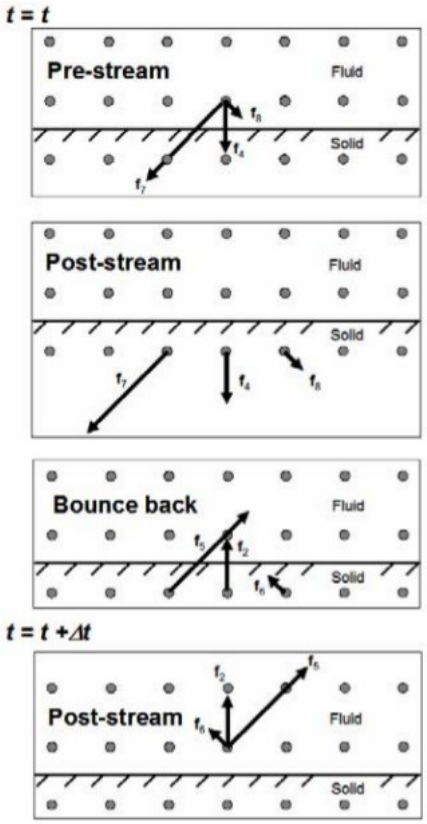


Figure 2: Bounceback movement of probabilities distribution functions f_a (Sukop and Thorne Jr., 2005)

4. IMPLEMENTATION

In this proposed algorithm for parallelizing the Lattice Boltzmann method, data-parallelism is applied to serial code, dividing the domain into smaller column blocks, as in Fig. 3. It was used block-cyclic partitioning, with block size equal to number of processes.

Steps of streaming, collision and boundary conditions can be performed independently, but after streaming it is needed to communicate left and right borders of each subdomain between processes in order to get the correct result. Since each subdomain has its own first and last column and they are exchanged after streaming, the whole domain need to be corrected, as Fig. 4 shows.

ISSN 2176-5480

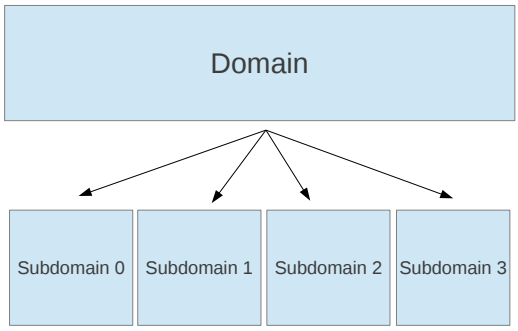


Figure 3: Domain division among processes

Collision and boundary conditions do not need any correction because, in these steps, lattice nodes do not access neighbour nodes, so calculations only depend on the node itself. This code uses a feature for saving 50% of memory in iterations by performing twice calculations to avoid temporary arrays; more details in (Latt, 2007). MPI functions are used to broadcast initial data to all processes, divide work for them, measure time and, on each iteration, exchange borders to right places. Output data is a set of vti extension files, which is an extension associated with ParaView VTK ImageData, for visualizing data, saved by each process individually and so also parallelized. The simulation can be watched in a vtk viewer program, such as ParaView.

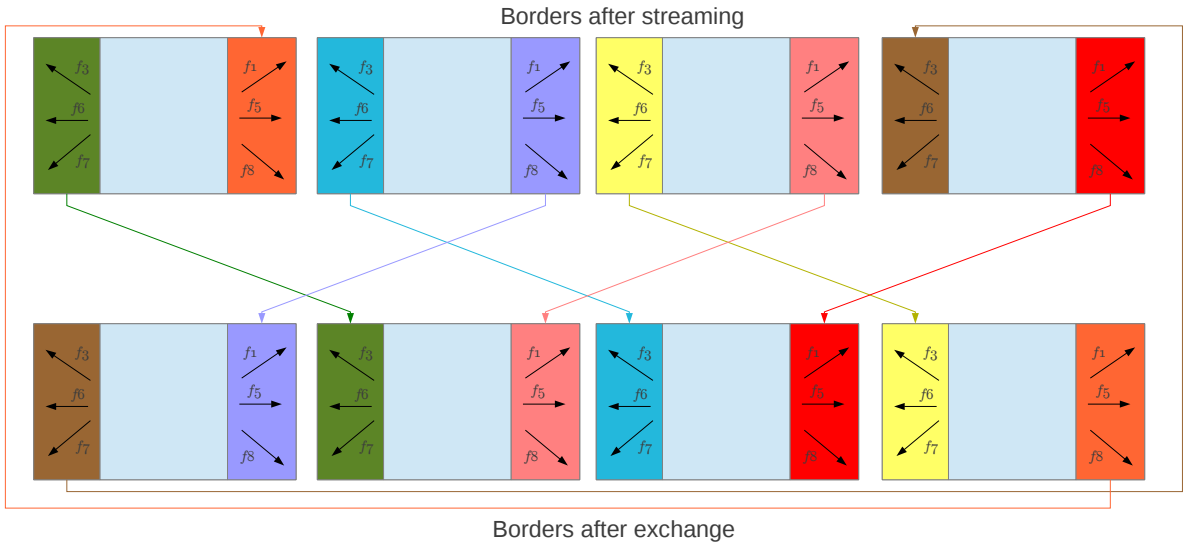
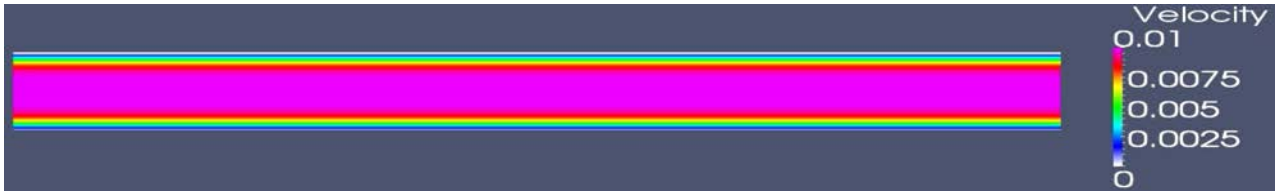


Figure 4: Subdomains after streaming (above) and after exchange of borders (below)

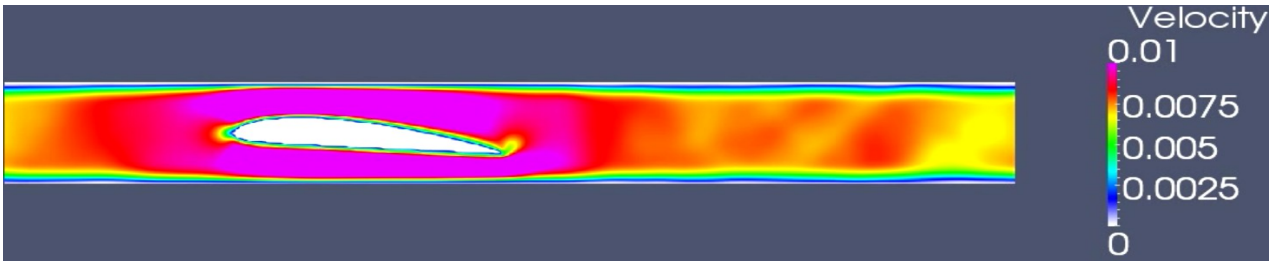
5. RESULTS

In order to validate the simulator, three cases of literature were tested: flow between parallel plates, flow through a cylinder and flow through an airplane airfoil (Fig. 5). As initial condition, a velocity of $0.01lu/lt$ was applied to every fluid node of domain. Periodic boundary was set in left and right edge of domain and in all cases there were plates in upper and lower edges. The microprocessor used to run these codes was Intel® Core^{TM} i7 CPU 950 at 3.07GHz $\times$ 8 (four cores, eight threads).



(a) Flow between parallel plates after 900 iterations

(b) Flow past a cylinder after 400 iterations



(c) Flow through an airfoil after 3000 iterations

Figure 5: Simulation cases

For performance comparison between serial and parallel code, the following sizes 512×64, 1024×128, 2048×256, 4096×512 and 8192×1024 lattice cells were used for each case in serial and in parallel code with two, four and eight processes, since the most recommended number of processes is the maximum number of processor's cores. The domain is described by a bitmap input file, that can be generated with a simple drawing software. Each pixel of the input bitmap file becomes a node of the lattice. 500 iterations were run and 100 output files as well as simulation time were saved. Figure 6 shows speedup (serial time divided by parallel time) for all cases.

From Fig. 6, it is possible to notice that the parallel code with two processes was around 20% faster than serial code. However, with four or eight processes, one can double performance with parallel code compared to serial run. There is almost no gain for eight processes because the microprocessor used in this test had four cores and eight threads, which confirms that the recommended number of processes is the same number of cores of CPU.

6. CONCLUSION

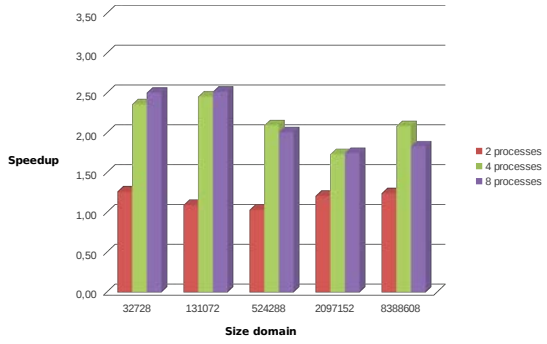
A 2D Lattice Boltzmann fluid simulator was implemented using MPI and its performance was compared to serial code. Three cases of literature in two dimensions were used: flow between parallel plates, flow past a cylinder and flow past an airfoil. Run times were measured and, with these results, speedups and efficiency were calculated in order to evaluate performance gain. Results showed that this parallel implementation using CPU with MPI is advantageous, and increases with number of processes, until the maximum number of cores of a microprocessor. MPI is worth to achieve performance gain, especially in distributed-memory systems, like clusters, when its use is essential.

Next step is to run this parallel code adapted for a cluster of CPUs using MPI, so that it is possible to simulate larger domains that requires more memory and achieve even higher speedups, as number of cores increases in a cluster. Another step is parallelize same code to run on Graphics Processing Units (GPUs) to increase the gain performance.

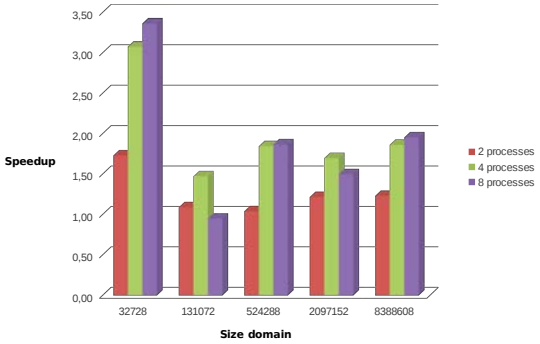
7. ACKNOWLEDGEMENTS

The authors are supported through grants from CAPES, CNPq grant $310474/2009-4$ and FAPESP grant $10/09717-7$.

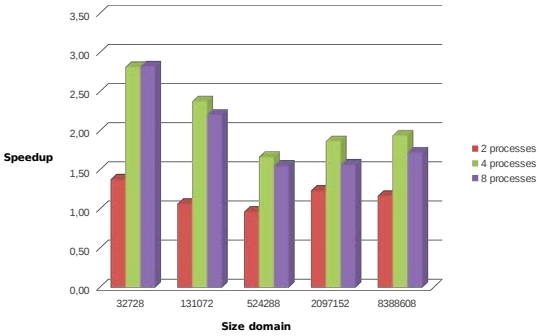
ISSN 2176-5480



(a) Speedup for parallel plates



(b) Speedup for cylinder



(c) Speedup for airfoil

Figure 6: Speedup for simulated cases

8. REFERENCES

Aidun, C.K. and Clausen, J.R., 2010. "Lattice-Boltzmann method for complex flows". *Annual review of fluid mechanics*, Vol. 42, pp. 439–472.

Chen, S., Doolen, G.D. and Eggert, K.G., 1994. "Lattice-Boltzmann, a versatile tool for multiphase fluid dynamics and other complicated flows". *Los Alamos Science*, Vol. 22, pp. 99–111.

Latt, J., 2007. "How to implement your DdQq dynamic with only q variables per node (insted of 2q)". Technical report, Tufts University Medford, USA.

Mohamad, A.A., 2011. *The Lattice Boltzmann equation for fluid dynamics and beyond*. Springer.

Oliveira, F.M.C. and Ferreira, L.O.S., 2012. "Simulation of the alcohol-oil mixture in a micromixer using the Lattice Boltzmann method on a GPU device". In *Proceedings of the 14th Brazilian Congress of Thermal Sciences and Engineering*. Rio de Janeito, Brazil.

Pacheco, P., 2011. *An introduction to parallel programming*. Elsevier.

Succi, S., 2001. *Lattice Boltzmann method: fundamentals and engineering applications with computer codes*. Numerical Mathematics and Scientific Computation. Oxford Science Publications.

Sukop, M.C. and Thorne Jr., D.T., 2005. *Lattice Boltzmann modeling: an introduction to geoscientists and engineers*. Springer.

Wolf-Gladrow, D.A., 2005. *Lattice-gas cellular automata and Lattice Boltzmann models - an introduction*, Vol. 1725. Springer.

Zhang, J., 2011. "Lattice Boltzmann method for microfluidics: models and applications". *Microfluid Nanofluid*, Vol. 10.

9. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.