Int J Parallel Prog (2009) 37:593–611
DOI 10.1007/s10766-009-0113-x

Parallel Lattice Boltzmann Method with Blocked Partitioning

Claudio Schepke · Nicolas Maillard · Philippe O. A. Navaux

Received: 15 April 2008 / Accepted: 9 June 2009 / Published online: 25 June 2009

Abstract This paper presents and discusses a blocked parallel implementation of bi- and three-dimensional versions of the Lattice Boltzmann Method. This method is used to represent and simulate fluid flows following a mesoscopic approach. Most traditional parallel implementations use simple data distribution strategies to parallelize the operations on the regular fluid data set. However, it is well known that block partitioning is usually better. Such a parallel implementation is discussed and its communication cost is established. Fluid flows simulations crossing a cavity have also been used as a real-world case study to evaluate our implementation. The presented results with our blocked implementation achieve a performance up to 31% better than non-blocked versions, for some data distributions. Thus, this work shows that blocked, parallel implementations can be efficiently used to reduce the parallel execution time of the method.

Keywords Cluster computing · Lattice Boltzmann method · High performance computing

This article was supported by CNPq.

C. Schepke · N. Maillard (✉) · P. O. A. Navaux
Grupo de Processamento Paralelo e Distribuído, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Caixa Postal 15.064, Porto Alegre, RS 91.501-970, Brazil
e-mail: nicolas@inf.ufrgs.br

C. Schepke
e-mail: cschepke@inf.ufrgs.br

P. O. A. Navaux
e-mail: navaux@inf.ufrgs.br

1 Introduction

Fluid Dynamics is an important technological research area. Through the study of the properties of liquids and gases it is possible to determine different kinds of physical phenomena [3]. Such phenomena are related with relevant and current problems like tsunami effects, hurricane simulations, droughts or flooding forecasts and the distribution of polluting particles in the atmosphere and in reservoirs of water. The evolution of computational systems made it possible to solve these problems in an efficient way through new simulation techniques. This area known as Computational Fluid Dynamics (CFD) is one of the most prominent areas in Scientific Computing [1,12].

Fluid simulations are frequently described through Navier-Stokes Equations [7]. Such equations relate different physical properties and forces that govern macroscopic fluid flows. In CFD these equations are discretized and solved through numerical methods. Although macroscopic simulations are viable in this way, only restricted cases generate accurate results [4]. Macroscopic approaches are generally insensible to microscopic dynamics. Therefore, some physical quantities that dynamically change in the real fluid flow will remain constant during all the simulation. A way to get over this limitation is to adopt mesoscopic approaches.

Instead of considering each particle individually, as in microscopic dynamics, mesoscopic approaches describe a physical system by a distribution function of the particles. An important mesoscopic simulation technique currently in use is the Lattice Boltzmann Method (LBM) [6,16,26,28]. The LBM is an iterative numerical method to model and to simulate fluid dynamics properties, where space, time and velocity are discrete [6]. The method enables the computational modeling of a large variety of problems, including fluid with multi-components, in one or more phases, with irregular boundary conditions and in complex geometries.

Depending on the dimension of the problems treated with the LBM, the amount of both memory and processing power that are required can become high. Hence, parallel strategies must be considered. Since the operations of the method involve only interactions with neighboring points of each lattice element, it is possible to use Data Parallelism [14]. Different solutions have been proposed with this view [8,10,19]. Most of them focus on the numerical results themselves or on the evaluation of the parallel performance of the computational systems. Much work on data parallelism has emphasized the performance obtained with blocked partitioning [9]. However, to the best of our knowledge, no blocked implementation of the LBM has been provided with a detailed analysis of its performance. Such an evaluation is very important because it can help determining the best regular data distributions for a LBM parallel implementation.

In this paper, we present two efficient, blocked partitioned, parallel implementations of one bi-dimensional and one three-dimensional version of the LBM. The communication cost is evaluated, which points at better efficiency than with mono-dimensional data distributions. We then proceed to an experimental evaluation of the parallel models. Our experiments show that blocked solutions provide better performance and scalability than non-blocked implementations.

The remainder of this article is divided in six sections. Section 2 presents some works related to the LBM and its parallelization, and the blocked partitioning concepts.

Section 3 develops the LBM, describing the equations, the boundary conditions and the algorithm. The blocked parallel implementation is detailed in Sect. 4, and its complexity in terms of communication is discussed in Sect. 5. In Sect. 6, the collected experimental results are presented for the simulation of a fluid flow. Finally, Sect. 7 provides a summary of the contributions and concludes.

2 Related Work

A frequent characteristic cited in the LBM literature is the possibility to parallelize it [26]: many authors consider parallelism as the best form to obtain fully performing implementations [20]. This section presents some of these parallel implementations, as well as some techniques that are used in Data Parallelism, that have been used in our approach.

2.1 Parallel Implementations of the LBM

There are many parallel implementations of the LBM [2,8,10,23]. Most of these works provide a parallel code to solve specific problems. Usually, these results are generally more concerned on the physical solution and its applications than on the parallel techniques. Exceptions are found in specific performance analysis of the method in computational systems, such as [5,24].

Carter and Oliker [5] presents some comparisons between implementations for vectorial and multi-computers architectures. The case studies are 2D and 3D implementations of the LBM for a turbulence problem. The results show that vectorial architectures are more scalable that multi-computers for the implementations tested.

A similar performance evaluation has been made in [24]. Three different applications involving nanotechnology and turbulence are tested on different high performance architectures (Hitachi SR8000-F1, SGI Altix, and NEC SX6), giving better results for the first. However, the cost-benefit of processing was better for the superscalar architecture.

In both previous cases, the applications are only used to compare the processing capacity of the method on different architectures, without evaluating the parallelization itself.

In terms of data partitioning, different solutions have been adopted for the LBM. The Orthogonal Recursive Bisection (ORB) method has been proposed for better load balance in massively parallel systems [19]. This approach has been tested, and yielded good performance [23]. However, such solutions are more indicated for irregular geometries. Another suggested technique is the Regular Domain Decomposition [8], but evaluations have only been made for cubic lattices. Moreover, the use of multi-dimensional partitioning has not been discussed in this work.

2.2 Blocked Parallel Algorithms

In many parallel applications using Data Parallelism, such as [8], the data distribution is performed in a simple way among all processors, generally by dividing equally the

data set in slices, each slice being allocated to one processor. However, this strategy does not explore well the data locality existing in the data set. The same situation occurs when the data set is divided cyclically among the processors. The use of graph decomposition is not a indicated decomposition technique for LBM because it work directly with discrete data. Particulary, we have interest in evaluate regular structures.

A more efficient way to distribute data among parallel processors is to use blocked partitioning, that is to divide in more than one dimension the data structures [9,21]. Blocked solutions are frequently adopted in numerical operations because this strategy enables a better memory access and cache use for the applications. It is also possible to reduce the communication costs and data dependencies in Distributed Memory Systems. For these reasons, they are often used in linear systems solvers [11,18].

Due to the importance of blocked partitioning in High Performance Computing, an efficient parallel implementation of the LBM should benefit from this technique. The next sections detail such a version of the LBM, but first its physical background is presented.

3 The Lattice Boltzmann Method (LBM)

The LBM is a mesoscopic method to describe the mechanics of a system of particles. The method is frequently adopted as an alternative technique for computational simulations of Fluid Dynamics, instead of using discrete Navier-Stokes equations solvers. In the LBM, space, time and velocity are considered discrete. Particles are represented as a regular mathematical structure called lattice. A lattice is formed by discrete points, each one with a fixed number of discrete displacement directions. At each iteration (of the discrete time), particles realize a space displacement among the lattice points. Through this structure it is possible to simulate the physical properties of fluid flows in a simple way.

Historically, the LBM is considered to be an evolution of the Lattice Gas Automata (LGA) [22] because it does not consider the individual motion of the particles and because it uses real values instead of logical values to simulate the particle displacement through the lattice. However, the LBM can also be derived from the Boltzmann Equation (BE) for diluted gases. Therefore, the method is considered as a discrete simulation technique for this equation too [17]. In both approaches, it is possible to determine the same Lattice Boltzmann Equation (LBE).

3.1 The Lattice Boltzmann Equation

The LBE can be developed from a kinetic equation of the distribution function of the particles, in terms of relaxation and propagation functions, as presented below.

- Relaxation:

$$f_i^{\text{new}}(x,t) - f_i(x,t) = \Omega(f_i(x,t)),\ (i = 1,\ldots,d) \tag{1}$$

- Propagation:

$$f_i(x+e,t+1) = f_i^{\text{new}}(x,t),\ (i=1,\ldots,d) \tag{2}$$

where f_i is the distribution function of the particle velocity in each one of the d lattice directions $i = 1,\ldots,d$, e is the space variation of x when the discrete time t changes to $t+1$ and Ω is a relaxation operator defined in (4). Joining Eq. 1 with Eq. 2 one gets:

$$f_i(x+e,t+1) = f_i(x,t) + \Omega(f_i(x,t)) \tag{3}$$

In the LBM, instead of considering all the collisions among the particles, a relaxation operator model called Lattice Bhatnagar-Gross-Krook (BGK) is used [13]. This relaxation operator is defined according to the mass and momentum conservation laws and is given by:

$$\Omega(f_i(x,t)) = -\frac{1}{\tau}(f_i(x,t) - f_q^{\text{eq}}(\rho,u)) \tag{4}$$

In this equation, τ represents a relaxation time scale, that controls the rate of equilibrium approximation, $f_q^{\text{eq}}(\rho,u)$ is the equilibrium distribution function and ρ and u are macroscopic values for density and velocity. The Eq. 5 defines the equilibrium distribution function $f_q^{\text{eq}}(\rho,u)$, where the value of w_i depends on the lattice geometry, e_i is the discrete velocity for each lattice direction and $c = \Delta x/\Delta t$.

$$f_q^{\text{eq}}(\rho,u) = \rho w_i\left[1 + \frac{3e_iu}{c^2} + \frac{9(e_iu)^2}{2c^4} - \frac{3u^2}{2c^2}\right] \tag{5}$$

The macroscopic values of density ρ and momentum ρu can be calculated from f, respectively as:

$$\rho = \sum_{i=1}^{d} f_i(x,t) = \sum_{i=1}^{d} f_i^{\text{eq}}(x,t), \tag{6}$$

$$\rho u = \sum_{i=1}^{d} e_i f_i(x,t) = \sum_{i=1}^{d} e_i^{\text{eq}} f_i(x,t), \tag{7}$$

3.2 Lattice Structures

This paper considers a bi-dimensional lattice structure, with 8 propagation directions (D2Q9) and a three-dimensional lattice structure, with 18 propagation directions (D3Q19), as shown in Fig. 1. To both cases the propagation can be null. Because this, is necessary to add one more propagation direction for both cases.

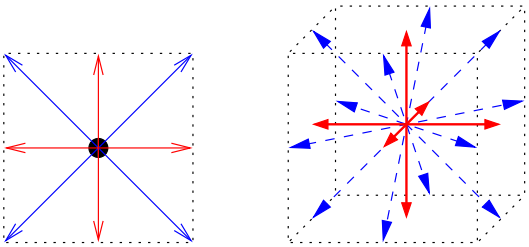

Fig. 1 D2Q9 e D3Q19 lattice geometry

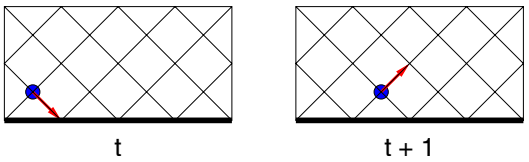

Fig. 2 Bounce-back mechanism

So, the nine possible directions of propagation of the D2Q9 model are:

- A static point (0, 0), where a particle has zero velocity. The value of w_i is 4/9.
- Four nearest directions $(+1, 0)$, $(-1, 0)$, $(0, -1)$ e $(0, +1)$, with unity velocity and $w_i = 1/9$.
- Four diagonal line neighbors $(1, 1)$, $(-1, 1)$, $(1, -1)$ e $(-1, -1)$, with velocity $\sqrt{2}$ and $w_i = 1/36$.

And the possible directions of the D3Q19 model are defined by:

- A static point at coordinate (0, 0, 0), where the particle has zero velocity. The value of w_i in this case is 1/3.
- Six nearest directions $(-1, 0, 0)$, $(+1, 0, 0)$, $(0, -1, 0)$, $(0, +1, 0)$, $(0, 0, -1)$ and $(0, 0, +1)$, with unity velocity and $w_i = 1/18$.
- Twelve diagonal line neighbors $(1, 1, 0)$, $(-1, 1, 0)$, $(1, -1, 0)$, $(-1, -1, 0)$, $(1, 0, 1)$, $(-1, 0, 1)$, $(1, 0, -1)$, $(-1, 0, -1)$, $(0, 1, 1)$, $(0, -1, 1)$, $(0, 1, -1)$ and $(0, -1, -1)$, with velocity $\sqrt{2}$ and $w_i = 1/36$.

3.3 Boundary Conditions

One of the most simple approaches to represent boundary conditions is to use a mechanism called **Bounce-back**. The idea of this mechanism consists in inverting the speed vectors directions when collisions occur against static points of the boundary:

$$f_i^{\text{out}}(x, t) = f_i^{\text{in}}(x, t) \tag{8}$$

Thus, all forces that are going out return back to the fluid. Figure 2 show the Bounce-back mechanism applied to a particle in two discrete consecutive time steps.

3.4 The Algorithm

The structure of the LBM algorithm is composed by the following operations:

1. Determine the initial conditions for all points of the lattice, to choose adequately the density ρ and speed e of the external cells (fluid boundary) and to define the relaxation time scale τ;
2. Calculate the macroscopic density ρ and speed u from the values of the variables of each lattice point, using the Eqs. 6 and 7;
3. Calculate the equilibrium function values using Eq. 5 and, then, get the result for each point of the relaxation function through the Eq. 4;
4. Use the equilibrium value to apply in the distribution function for each lattice point, as shown by Eq. 3;
5. Propagate the particles distribution to all neighboring cells;
6. Modify the local lattice points distribution to satisfy the boundary conditions using Eq. 8;
7. Return to step 2 while the execution time or the iterations number is less than the maximum number of steps determined.

4 Blocked Parallel Lattice Boltzmann Method

In this work, a 2D and a 3D LBM have been parallelized, based on two different given lattices (see Fig. 1). The objective of these implementations was to obtain some macroscopic values, like velocity and pressure, for a fluid flowing through pipes with obstacles. The LBM code was written in C for a Distributed Memory System, using a Single Program Multiple Data (SPMD) parallel programming model [27]. In a SPMD approach, a unique program is executed in each process, operating over distinct data regions.

4.1 Implementation Issues

In the parallel version of the algorithm presented in 3.4, each process runs its operations on a predetermined region of the lattice. The data are divided according to the number of partitions defined for each axis direction (x, y and z) and equally distributed among the processes.

The main data-structure of the program defines the lattice and the velocity e of each one of its points, as follows:

```
//lattice structure
typedef struct {
  long int lx_total, ly_total, lz_total;
  long int lx_first, ly_first, lz_first;
  long int lx_last, ly_last, lz_last;
  bool ***obst; //obstacle 3D array
  double ****speed; //velocity
} s_lattice;
```

This data-structure is used to define the sub-lattice local to each one of the SPMD processes considering the D3Q19 lattice model. (For the D2Q9 model the data-structure is similar, but more simple). The three first fields `lx_total`, `ly_total` and

lz_total define the size of the global lattice ($N =$ lx_total $\times$ ly_total $\times$ lz_total). Each process owns and stores only the local sub-lattice ranging from (lx_first to lx_last-1) $\times$ (ly_first to ly_last-1) $\times$ (lz_first to lz_last-1). Thus, the global lattice is blocked distributed between all the p SPMD processes. Finally, the data-structure informs the localization of the obstacles points (as a three dimensional array) and the speed, in direction $i = 1, \ldots, d$, of each one of the N/p points of the local sub-lattice (implemented as a four-dimensional array). We not use sparse data-structures for the speed because few points are null.

At the end of each loop iteration, data exchanges are made between the neighbor processors: boundary data from each sub-lattice of each processor are sent to the neighbors processors, since each processor operate with a sub-lattice. This communication consist in data transmissions between orthogonal and diagonal sub-lattices, according to the particle distribution. The Sect. 5 describes the communication scheme and its complexity.

4.2 Parallelization with MPI

The parallel implementation has been performed with Message Passing Interface (MPI) [15,25]. MPI is the main standard for parallel computing. It provides primitives and data-structures for point-to-point and collective communication, as well as for synchronization between the processes that participate to the computation. MPI is both efficient and portable.

The functionalities of MPI that have been used for the LBM parallel implementation include Cartesian mapping, synchronous, asynchronous and collective communication. A Cartesian communicator structure is used and defined at the beginning of the computation, by a call to `MPI_Cart_create()`. This function defines the total number of coordinate dimensions adopted in the data partition and the number of divisions for each one. Then, each MPI process is associated to this Cartesian communicator using the `MPI_Comm_rank()` function and mapped by `MPI_Cart_coords()`. Thus, each process can be identified by a Cartesian position in a grid. Depending of this Cartesian position, each process determines the blocked sub-lattice that it will compute, as previously presented (see 4.1).

For each one of the d directions, the velocity of the particles that should migrate from one sub-lattice to a neighbor must be transmitted to the neighbor in that specific direction. The velocities are buffered, in order to be efficiently sent. The communication among the processes due to the data at the border of each cell is performed with asynchronous operations `MPI_Isend()`. On the receiver side, the processes use blocking receives operations `MPI_Recv()`, because the data is required on the computation.

Collective and synchronous MPI communication have been used to calculate the global mean velocity of the flow in each iteration. The mean velocity is defined for all points of axis y and z for a fixed x-axis element. Thus, the calculation is made only by processes where the selected region is present. All partial values are sent to a root process through `MPI_Send()` that receives it by `MPI_Recv()`. We use `MPI_Send()` and `MPI_Recv()` because only some processors are involved in this operation. The

root process makes the sum and distributes the resulting value through the collective function `MPI_Bcast()` to all processes. This way, all the processes know when the stopping criterion is reached.

5 Communication Complexity of the Blocked Parallel Version

In order to evaluate the communication cost of a blocked partitioned, parallel version of the LBM, the time to send a message of size N is modeled as:

$$\tau = L + Ng = L + N/\sigma, \tag{9}$$

where L is the latency and g is the inverse of the network throughput σ.

5.1 Bi-dimensional Model

In the D2Q9 lattice model there are 9 directions of particle propagation: 1 static, 4 orthogonal and 4 diagonal. These propagations imply communication, yet different directions can be grouped, due to the Cartesian 2D decomposition of the lattice. As can be seen in Fig. 1, each sub-lattice owned by a processor of rank (p, q) in the Cartesian grid will have to communicate 3 propagations to $(p-1, q)$, $(p+1, q)$, $(p, q-1)$ and $(p, q+1)$ (see Fig. 3 for an example in the case of the "vertical" communication).

Let us call $N = N_x \times N_y$ the dimensions of the lattice, and $P \times Q$ the size of the Cartesian 2D grid of processors. Then, each sub-lattice on each processor contains roughly $n_x \times n_y$ points, with:

$$n_x = N_x/P, \tag{10}$$
$$n_y = N_y/Q, \tag{11}$$

and the communication penalty is:

$$C_o = 4L + \frac{6(n_x + n_y)}{\sigma}. \tag{12}$$

considering four communications (one in each direction) of three data information (e_i) for each processor.

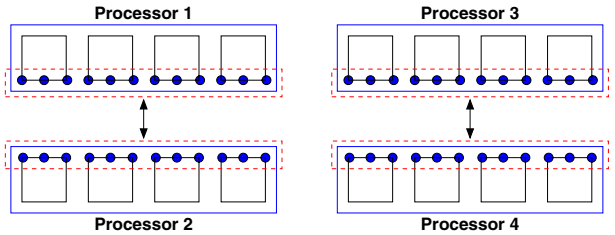

Fig. 3 Example of communication between processors in the D2Q9 model

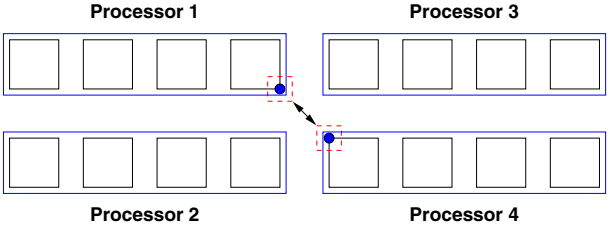

Fig. 4 Example of communication of diagonal points in the D2Q9 model

However, this does not include the necessary communication of the diagonal extreme points of each sub-lattice (see Fig. 4).

In this example, the Processor 1 send the information of the diagonal propagation of its more extreme sub-lattice point to its diagonal neighbor, which is Processor 4. Only the communication of these two processors are shown for illustration, but the same data emission occurs to the other three diagonals. Therefore, each processor needs to make four sends of only one value (each are of cost $L + 1/\sigma$) to its four diagonal processors, which leads to the communication cost:

$$C_d = 4(L + (1/\sigma)) \tag{13}$$

Thus, the total communication cost is given by $C_o + C_d$:

$$C = 8L + \frac{2}{\sigma}\left(3\left(\frac{N_x}{P} + \frac{N_y}{Q}\right) + 2\right). \tag{14}$$

Notice that this equation gives an insight on the interest of a block partition: the communication cost is dominated by $N_x/P + N_y/Q$, i.e. by a term that is equivalent to the square root of the total number of points N divided by P, if the geometry is squared. In a mono-dimensional partitioning, it would be dominated by a linear function of N.

5.2 Three-dimensional Model

The communication costs of the D3Q19 lattice are similar to the bi-dimensional case. In the D3Q19 model there are 18 different directions of particle propagations, plus the static position. These 18 directions can be grouped, to reach the 6 orthogonal neighbors of each sub-lattice of a given processor of rank (p, q, r) in the 3D Cartesian grid. Each of these communications group five different propagation directions: one orthogonal direction and four diagonals (see Fig. 1 and also Fig. 5 for an illustration of the communication in one of the orthogonal directions.

Noting $N = N_x \times N_y \times N_z$ the number of points in the lattice, and with $P \times Q \times R$ processors, each one of them owns roughly $n_x \times n_y \times n_z = N_x \times N_y \times N_z/(PQR)$ points of the lattice.

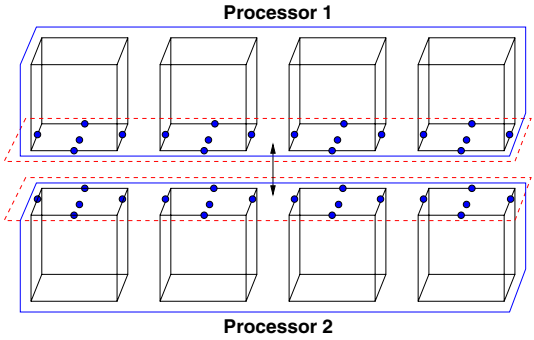

Fig. 5 Example of communication between processors in the D3Q19 model

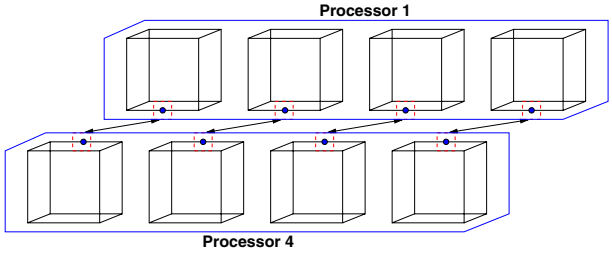

Fig. 6 Example of communication of diagonal points in the D3Q19 model

Therefore, the communication cost (to 5 data information e_i) due to the orthogonal communication (6 directions) is:

$$C_o = 6L + \frac{2}{\sigma}\left(5(n_y n_z + n_x n_z + n_x n_y)\right). \tag{15}$$

Yet again, a few diagonal communication must occur: an example of diagonal communication is shown in Fig. 6. Once more, only two processors are used for the illustration, showing the data exchange only along one diagonal of the propagation. However, each one of the 12 diagonal directions of the extreme points of the sub-lattice need to be sent to the neighbor processors (four to each axis x, y and z).

Then, when the lattice is partitioned in a 3D geometry, the communication cost for the diagonal elements is:

$$C_d = 12L + \frac{4}{\sigma}\left(n_x + n_y + n_z\right). \tag{16}$$

The total communication costs is obtained by $C_o + C_d$:

$$\begin{aligned} C &= 18L + \frac{10W + 4Z}{\sigma}, \\ W &= n_y n_z + n_x n_z + n_x n_y, \\ Z &= n_x + n_y + n_z. \end{aligned} \tag{17}$$

This time, if the geometry is totally squared, the global communication cost will be dominated by $N^{\frac{2}{3}}$ versus N with a mono-dimensional partition.

In both models, this theoretical analysis points at the block partitioning as a better solution than the mono-dimensional partitioning. The next section presents an experimental case study that confirms this result.

6 Performance Analysis

In order to evaluate the parallel implementation of the LBM, three case-studies are presented. These case-studies are based on simulations of a fluid flow through a canal with obstacles.

6.1 Case Studies

The first case study uses a bi-dimensional lattice structure with dimension $N = 512 \times 512$ points, that represents a pipe. Five obstacles forming a barrier are distributed cyclically following the x axis, as can be seen in Fig. 7. In the same figure, one can also observe the direction of the velocity and the two opened faces of the canal where the fluid circulates.

The second and third case studies have three-dimensional lattice structures. The second case has $N = 128 \times 128 \times 128$ points. The other case study has a dimension of $N = 256 \times 256 \times 256$ points. Both have a cubic structure to simplify the considerations about data distributions strategies. The obstacles in both three-dimensional cases are formed by a rectangular set of elements placed in the canal at the first third of the axis x, as represented in Fig. 8. In this figure one can also observe the direction of the velocity and the two opened faces of the cube where the fluid circulates.

The tests with the bi-dimensional lattice case study model were made with 150,000 iterations of the main loop of the method. For the two 3D models, the difference of the mean velocity between two consecutive iterations, measured on the points in the middle vertical plane, has been adopted as stopping criterion with tolerance $\varepsilon = 10^{-4}$. To reach this criterion, 447 iterations have been necessary for the $128 \times 128 \times 128$ lattice and 495 iterations for the other one.

An illustration of the velocity and pressure distribution after 150,000 iterations of the bi-dimensional case study can be seen in Fig. 9. A similar illustration, projected on the middle vertical plane $x = 64$ of the $128 \times 128 \times 128$ lattice is presented in Fig. 10.

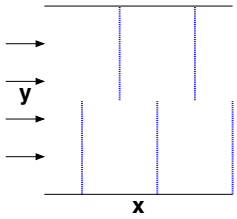

Fig. 7 The obstacle in the bi-dimensional case

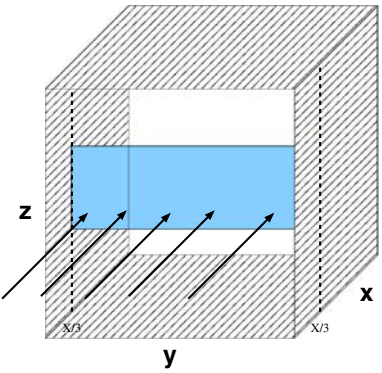

Fig. 8 The obstacle in the channel

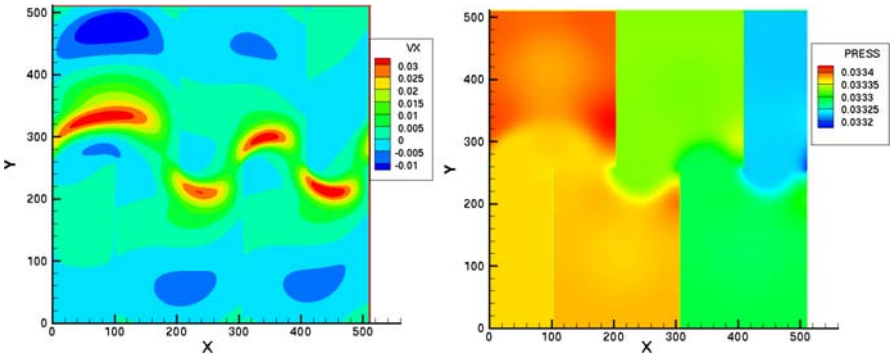

Fig. 9 Velocity and pressure distribution after 150,000 iterations, 2D model

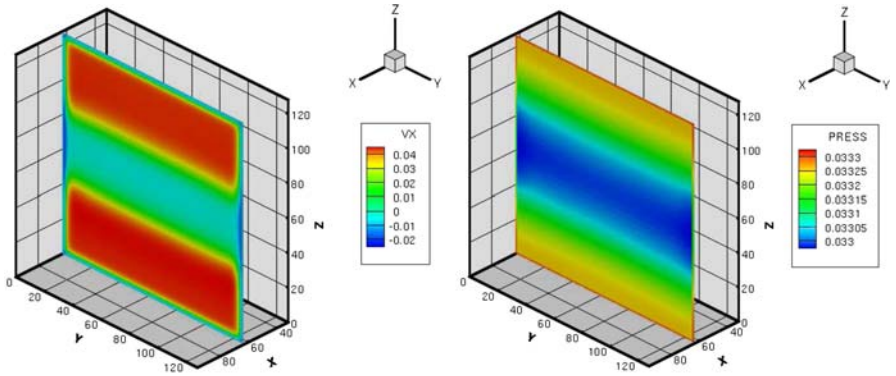

Fig. 10 Velocity and pressure distribution in the vertical plane $x = 64$, 3D model

6.2 Simulation Environment

All measurements have been made on the cluster Labtec, at the Institute of Informatics of the Federal University of Rio Grande do Sul. This cluster is composed by 20 dual nodes Pentium III, 1.266 Ghz with 512 MB of RAM memory, interconnected by a

Fast Ethernet network. In addition, MPICH version 1.2.7 implementation has been used for all the tests.

The computational results presented are based on 10 executions. After removing the best and worst runtimes, we took the average of the remaining 8 executions. The standard deviation of the values that compose the averages was below 2%. The execution time measures the main loop of the method using the `MPI_Wtime()` function of the MPI library. Using this methodology, it was possible to evaluate the scalability and the performance of the different adopted partitioning strategies.

6.3 Results with the Bi-dimensional Model

6.3.1 Scalability

The speed-up of the mono-dimensional partitioning of the lattice with 512×512 points is shown in Fig. 11. In this figure the results of line and column partitioning are compared with the ideal speed-up. The line partitioning presents better results than the column partitioning because the first partition strategy has better memory locality, making better use of cache when less data need to be processed.

The Table 1 presents the best execution time for some mono- and bi-dimension partitioning strategies. The table also shows the percentage relation between the best result for each number of processors using mono- and bi-dimension partitioning. The conclusion is that bi-dimensional strategies have a global execution time that is between 5.6 and 8.2% lower than with mono-dimensional partitioning, when the number of processors is higher than 25.

6.3.2 Execution Time

In terms of execution time, using 40 processors and different partition strategies, better results are obtained with blocked bi-dimensional partitioning, as can be see in Fig. 12.

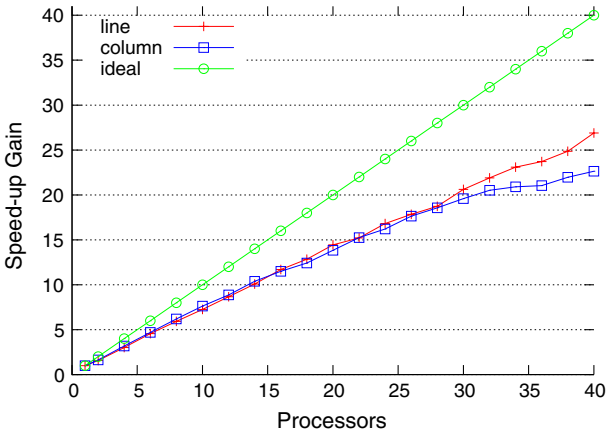

Fig. 11 Speed-up of the mono-dimensional partitioning of the D2Q9 case study

Table 1 Best execution time using 27, 32, 36 and 40 processors dividing the D2Q9 case study in one and two dimensions

Distribution	Time	% of reduction time in relation of the mono dimensional division
25×1	5137.72	–
5×5	4739.85	7.74
32×1	4037.49	–
4×8	3808.57	5.67
36×1	3738.54	–
9×4	3430.8	8.23
40×1	3418.07	–
5×8	3152.42	7.77

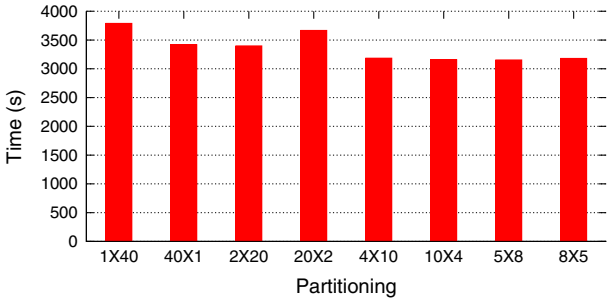

Fig. 12 Execution time using 40 processors for the D2Q9 case study

These results show that the more quadratic the block size is, the better the execution time.

6.4 Results with the Three-dimensional Model

6.4.1 Scalability

The scalability of the parallel implementation of the LBM has been evaluated using the lattice with $128 \times 128 \times 128$ elements. Figure 13 shows the speed-up obtained for the best mono-partitioning results (column partitioning) and for the best blocked partitioning strategy.

One observes that blocked solutions are more scalable than mono-partitioning implementations. Using 40 processors, the speed-up for the mono-partitioning strategy has been approximately 18. With the blocked strategy, it has achieved 25. Besides, mono-partitioning presents a fast stabilization of the speed-up factor, while the blocked partitioning tends to maintain a continuously increasing speed-up.

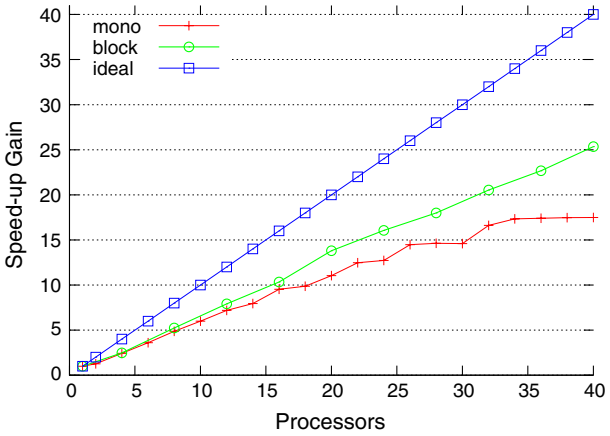

Fig. 13 Speed-up for mono- and blocked partitioning of the D3Q19 case study

6.4.2 Execution Time

Execution times have been obtained for all the possible partitions of the data between the 40 processors of the 20 nodes cluster.

In Table 2, the execution times of the mono-partitioning strategy for both case studies are presented. The values in the distribution field ($40 \times 1 \times 1$, for example) indicate the number of cells in the partition in each one of the three x, y and z dimensions. In each one of the three mono-dimensional distributions, only one of the three axis has been distributed between the 40 CPUs of the cluster. For both sizes of the lattice, the runtime has been significantly higher when the x axis has been partitioned.

Figures 14 and 15 show, respectively, the execution time for two- and three-dimensional partitions of the lattices with $128 \times 128 \times 128$ and $256 \times 256 \times 256$ elements. The values are relative to the execution time obtained for the $40 \times 1 \times 1$ partition strategy. This was the slowest one in both case studies. Thus, this normalization facilitates the evaluation of the runtime gain.

The results demonstrate the advantages of using blocked partitioning. For the lattice with $128 \times 128 \times 128$ points, the performance has been up to 38% better for blocked partitioning than for non-blocked partitioning. For the lattice with $256 \times 256 \times 256$ points the gain in runtime reaches 22 and 18%, respectively, for two-dimensional and three-dimensional partitioning.

Table 2 Execution time using mono-dimensional partitioning for the D3Q19 case studies

Distribution	$128 \times 128 \times 128$ (s)	$256 \times 256 \times 256$ (s)
$40 \times 1 \times 1$	352.4	2399.9
$1 \times 40 \times 1$	318.6	2281.3
$1 \times 1 \times 40$	322.8	2264.9

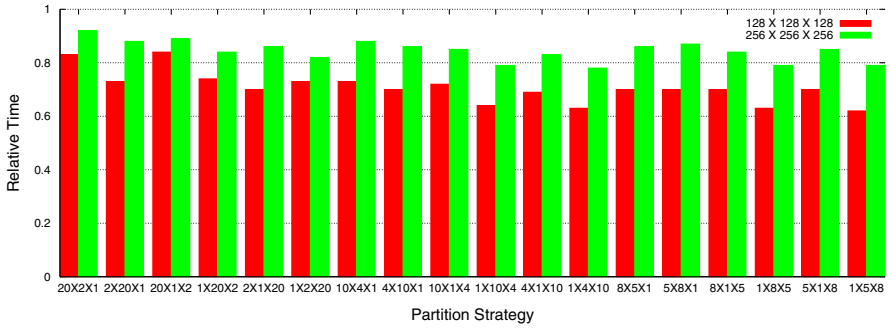

Fig. 14 Relative execution time using 2D blocked partitioning for the D3Q19 case studies

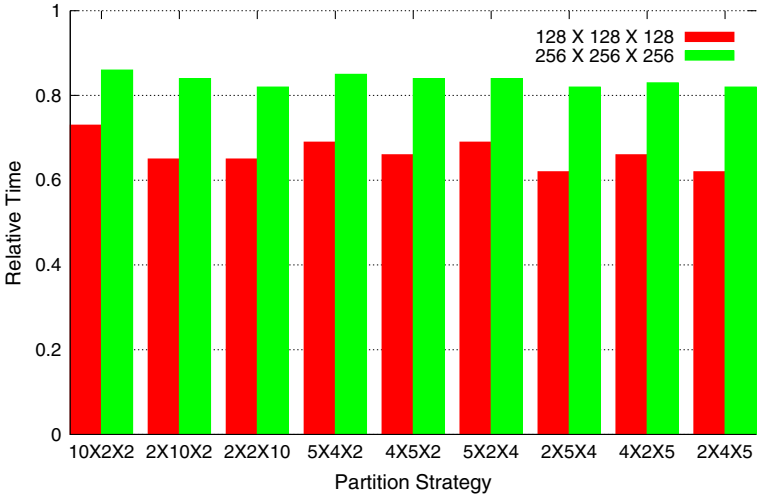

Fig. 15 Relative execution time using 3D blocked partitioning for the D3Q19 case studies

Considering now the best case for each kind of data distribution, the performance for the lattice with $128 \times 128 \times 128$ points reaches up to 31% in relation to the best mono-partitioning approach. For the lattice with $256 \times 256 \times 256$ points, analyzing only the best results in all data distributions, the increases of performance for two- and three-partitioning approaches, when compared to the mono-partitioning approach, are respectively, 17 and 14%.

In all results, one observes that when the x axis is partitioned, the execution time is higher. This occurs because the mean velocity used as a stop criterion must be calculated in parallel when this axis is divided. Therefore, there are more communication and synchronization operations in this case. This is the main reason why three-dimensional partitioning presents similar results as two-dimensional partitioning.

7 Conclusion

This article has presented a parallel version of the LBM with different partitioning strategies of the data, both for a bi-dimensional geometry and for a three-dimensional one. The use of efficient blocked data distributions clearly leads to an increased performance in parallel implementations, as shown by a simple complexity analysis and by the experimental results. Blocked implementations show to be both better performing and more scalable than a mono-dimensional parallelization for Regular Domain Decompositions.

One of the points that has not been considered in the definition of the blocks are the obstacles. Yet, their localization strongly influences the distribution of the velocities in the fluid. Typically, if the obstacles were parallel to the plan $x - y$, the blocked distribution should take x (and y) into consideration.

Another point is the adaptation of the size of the blocks to irregular loads during the computation. This implementation has used the same block size on each one of the cluster's node. Since the cluster is homogeneous, this choice is close to optimal. In a heterogeneous context, or if the lattice is not uniform, blocks of different sizes should be used.

In this work we used only one sub-lattice in each processor. So, it was used the maximum of local processing, reducing the cost of communications in the run time. We have interest to analyze what happens with the running time if more than one sub-domain is implemented by processor since the number of times required for data exchanging amoung the processes increase.

References

1. Anderson, J.D.: Computational Fluid Dynamics. McGraw-Hill, New York (1995)
2. Artoli, A., Hoekstra, A., Sloot, P.: Optimizing Lattice Boltzmann simulations for unsteady flows. Comput. Fluids **35**(2), 227–240 (2005)
3. Batchelor, G.K.: An Introduction to Fluid Dynamics. Cambridge University Press, Cambridge, MA (1987)
4. Buick, JM.: Lattice Boltzmann methods in interfacial wave modelling. Ph.D. Dissertation, University of Edinburgh, Fluid Dynamics Group, Feb. 1997
5. Carter, J., Oliker, L.: Performance evaluation of Lattice-Boltzmann magnetohydrodynamics simulations on modern parallel vector systems. Lawrence Berkeley National Laboratory, Paper LBNL-59340, Jan. 2006
6. Chen, S., Doolen, G.D.: Lattice Boltzmann method for fluid flows. Annu. Rev. Fluid Mech. **30**, 329–364 (1998)
7. Chung, T.: Computational Fluid Dynamics. Cambridge University Press, Cambridge (2003)
8. Desplat, J.-C., Pagonabarraga, I., Bladon, P.: Ludwig: a parallel Lattice-Boltzmann code for complex fluids. Comput. Phys. Commun. **134**(3), 273–290 (2001)
9. Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A., (eds.): The Sourcebook of Parallel Computing. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier, Nov. 2002
10. Dupuis, A.: From a Lattice Boltzmann model to a parallel and reusable implementation of a virtual river. Ph.D. Dissertation, University of Geneva, CUI, Computer Science Departement, University of Geneva (2002)
11. Elmroth, E., Gustavson, F., Jonsson, I., Kågström, B.: Recursive blocked algorithms and hybrid data structures for dense matrix library software. SIAM Rev. **46**(1), 3–45 (2004)
12. Ferziger, J.H., Peric, M.: Computational Methods for Fluid Dynamics. Springer-Verlag, London (2002)

13. Flekkøy, E.G.: Lattice Bhatnagar-Gross-Krook models for miscible fluids. Phys. Rev. E **47**(6), 4247–4257 (1993)
14. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley, Reading, MA (1995)
15. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message Passing Interface. MIT Press, Cambridge, Massachusetts, USA (1994)
16. He, X., Luo, L.-S.: Lattice Boltzmann model for the incompressible Navier-Stokes equation. J. Stat. Phys. **88**, 927–944 (1997)
17. He, X., Luo, L.-S.: Theory of the Lattice Boltzmann equation: from Boltzmann equation to Lattice Boltzmann equation. Phys. Rev. E **56**, 6811–6817 (1997)
18. Jonsson, I., Kågström, B.: Recursive blocked algorithms for solving triangular systems—Part I: one-sided and coupled Sylvester-type matrix equations. ACM Trans. Math. Softw. **28**(4), 392–415 (2002)
19. Kandhai, D., Koponen, A., Hoekstra, A., amd, K.M., Timonen, J., Sloot, P.: Lattice-Boltzmann hydrodynamics on parallel systems. Comput. Phys. Commun. **111**, 14–26 (1998)
20. Körner, C., Pohl, T., Rüde, U., Thürey, N., and Zeiser, T.: Parallel Lattice Boltzmann methods for CFD applications. In: Bruaset, A., Tveito, A. (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, vol. 51 of Lecture Notes for Computational Science and Engineering, Chap. 5, pp. 439–465. Springer-Verlag, London (2005). ISBN 3-540-29076-1
21. Lam, M.D., Rothberg, E.E., Wolf, M.E.: The cache performance and optimizations of blocked algorithms. In ASPLOS-IV: Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 63–74. ACM Press, New York, NY, USA (1991)
22. McNamara, G., Zanetti, G.: Use of the Boltzmann equation to simulate Lattice-Gas Automata. Phys. Rev. Lett. **61**, 2332–2335 (1988)
23. Pan, C., Prins, J.F., Miller, C.T.: A high-performance Lattice Boltzmann implementation to model flow in porous media. Comput. Phys. Commun. **158**(2), 89–105 (2004)
24. Pohl, T., Thürey, N., Deserno, F., Rüde, U., Lammers, P., Wellein, G., Zeiser, T.: Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures. Supercomputing (2004)
25. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: the Complete Reference, vol. 1 and 2. The MIT Press, Cambridge, MA (1998)
26. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, New York, USA (2001)
27. Wilkinson, B., Allen, M.: Parallel Programming: Using Networked Workstations and Parallel Computers. Prentice Hall, USA (1998)
28. Wolf-Gladrow, D.A.: Lattice-Gas Cellular Automata and Lattice Boltzmann Models: an Introduction. Springer, Berlin (2000)