A Grid-enabled Lattice-Boltzmann-based modelling system

Gérard Dethier[1], Cyril Briquet[1], Pierre Marchot[2] and P.A. de Marneffe[1]

[1] Department of Electrical Engineering and Computer Science
[2] Department of Chemical Engineering
University of Liège
B-4000 Liège

Abstract. Lattice-Boltzmann (LB) methods are a well-known technique in the context of computational fluid dynamics. By nature, they can easily be parallelized but their adaptation to the Grid environment is not trivial due to hardware heterogeneity (CPU, memory...) in a Grid. A load balancing method to dynamically handle the differences in terms of CPU number and power among the machines of a Grid is presented. The CPU power is dynamically estimated using a benchmark. An estimation method of execution time is also given.

1 Introduction

In the context of complex fluid flow simulations, standard techniques include those from computational fluid dynamics (CFD), one branch of fluid mechanics. The base of these techniques involves the solving of the Navier-Stokes equations using numerical methods and algorithms.

Lattice Boltzmann (LB) methods constitute one family among these techniques. They present several advantages like dealing with complex boundaries (i.e. solids with complex geometries like porous media...), incorporating microscopic interactions and being easily parallelizable.

However, these algorithms need a lot of computational and memory resources. They need very powerful machines or computational grids. The latter solution has the advantage of being economically more interesting. In addition to that, sufficiently large machines do not exist for extreme cases. However, adapting LB-based codes to Grid computing is not simple. The main obstacle is the high level of heterogeneity in terms of hardware, software and temporal availability of the machines that are to be used.

This paper presents LaBoGrid, a modelling system under development which is based on LB-methods. It is currently able to take into account the CPUs heterogeneity of the Grid machines (i.e. their number and their speed).

Here after, the concepts of Grid computing and the LB methods are outlined. A load balancing method and the LaBoGrid architecture are introduced. An estimation method of LB job execution time is given and, finally, results and conclusions will be presented.

1.1 Grid computing

Grid computing can be defined as "*coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations*" [1]. Another widely accepted definition comes from Ian Foster's *What is the Grid?* paper [2]. "*A Grid is a system that :*

1. coordinates resources that are not subject to centralized control,
2. using standard, open, general-purpose protocols and interfaces,
3. to deliver nontrivial qualities of service."

In practice, a Grid User submits a job composed of tasks to the Grid. The tasks will be distributed among the available resources.

In our context, a resource is a machine of the Grid characterized by its CPU(s) and memory. The machines of a Grid do not necessarily have the same amount of memory and cores. A resource can therefore be single- or multi-core (in the remainder of this paper, we consider the terms "core" and "CPU" as equivalent). Moreover, the CPUs don't necessarily exhibit the same speed and could be based on different architectures.

1.2 LB Methods

The LB methods model the displacements of fictitious particles along given velocity vectors on discrete lattices in a discrete time base. This means that space, velocities (i.e. velocity vectors) and time have been discretized. The particles are displaced from one site (a node of the lattice) to another neighbouring site.

The neighbourhood of a site is defined by the velocity vectors that a particle is allowed to follow. In our case, 3D fluids are modelled. Each site has 19 neighbours. This situation is illustrated on Figure 1. This lattice model is named D3Q19 following the generic naming scheme DiQj where i is the number of dimensions of the lattice and j the number of neighbours for each site. From now on, the lattices considered in the remainder of this paper will all be D3Q19.

The state of each site is given by a vector of real values associated to each possible velocity vector. Each value represents the proportion of particles (usually called field in the LB context) moving along a given velocity. In this paper, the state of a site consists of 19 real values. The time evolution of the sites is given by the two following differential equations [6] according to a widely used formalism [3].

$$\hat{f}_i(\boldsymbol{x} + \boldsymbol{v_i}\delta t, t) = f_i(\boldsymbol{x}, t) \tag{1}$$

$$f_i(\boldsymbol{x}, t+\delta t) = \hat{f}_i(\boldsymbol{x}, t) + \frac{1}{\tau}(f_i^{eq}(\boldsymbol{x}, t) - \hat{f}_i(\boldsymbol{x}, t)) \tag{2}$$

where $\boldsymbol{v_i}$ is the i-th velocity vector and δt the time step. The $f_i^{eq}(\boldsymbol{x}, t)$ is the equilibrium state calculated in function of $\hat{f}_i(\boldsymbol{x}, t)$. τ is the relaxation coefficient.

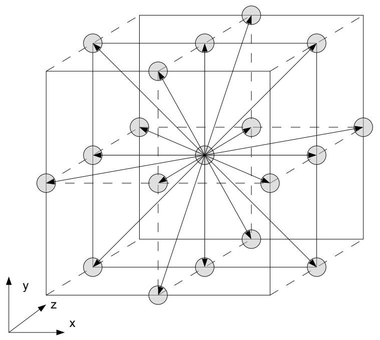

Fig. 1. Neighbourhood of a site in the D3Q19 model. The grey circles indicate the 19 neighbours. The site itself is also considered as a neighbour.

```
"Initialise the state of the fluid";
$i := 0$;
do $i < iterationsCount$ →
    "Translate particles proportions";
    "Collide particles proportions";
    $i := i + 1$
od
```

Table 1. LB algorithm

Equation 1 is the propagation step explaining how a site "exchanges" data with its neighbours. Equation 2 is the collision step explaining how the new state of a given site is calculated in function of its incoming data.

Table 1 shows a generic LB algorithm. The "*Translate particles proportions*" step implements Equation 1. The "*Collide particles proportions*" step implements Equation 2.

In the "*Translate particles proportions*" step, the sites on the boundaries of the lattice have no neighbours following some velocity vectors. The effect is that these sites have outgoing data but no incoming. The most common workaround is to use circular boundary conditions. What goes out from one side comes in to the opposite side and vice versa.

In general, the flow of a fluid through a given structure like a pipe or a porous medium is modelled. This structure is generally represented by a bitmap associating to each site its nature (solid or not). The nature of each site is taken into account in the collision step. If a site is of solid nature, the "bounce-back" [4] is applied instead of the equation 2. In the bounce-back, the values of opposite velocities are exchanged.

The flow of the fluid is driven by input and output pressure differences [5]. This method implies a modification of the $\hat{f}_i$ (see equation 1) at "input" and "output" of the lattice before the collision. For a given modelling, the flow will follow one direction (x, y or z). The input of the lattice is the plane at position

0 in the direction of the flow. The output is the opposite plane depending on the flow direction. Actually, these special planes don't need incoming data any more because the incoming data are calculated by the pressure difference application. This implies that the circular boundary conditions can be ignored for these planes. We can safely "ignore" the outgoing values.

1.3 Adapting LB methods to the Grid

The LB algorithm previously described is decomposed into a set of tasks, i.e. a job. Only one task my be assigned to each resource. As resources can have several available CPUs, each task will instantiate a number of threads equal to the number of available CPUs. The initial lattice is subdivided into sub-lattices. An example of a lattice decomposed into several sub-lattices can be seen in Figure 2. The sub-lattices will be processed by the threads of the tasks.

If the sub-lattices are all cuboids and if a sub-lattice shares a face or an edge with only one other sub-lattice per face or edge, the neighbourhood of a sub-lattice has the same definition as a lattice site's neighbourhood (see Figure 1). A sub-lattice has therefore 19 neighbours, which are not necessarily different and can include the sub-lattice itself. In Figure 3, the neighbourhood of 2 sub-lattices is showed. Only one communication channel between these two sub-lattices is needed.

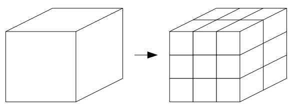

Fig. 2. 3D lattice decomposed into 18 sub-lattices.

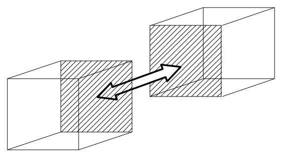

Fig. 3. 2 sub-lattices and their neighbourhood.

The step "*Translate particles proportions*" of the algorithm presented in Table 1 must be adapted in order to take into account the data exchange between sub-lattices. Indeed, the data going out of a sub-lattice at each LB time iteration must be transmitted to the neighbouring sub-lattices.

The step "*Collide particles proportions*" of the algorithm presented in Table 1 must also be adapted. Before the collision operator can be applied on the boundaries of a sub-lattice, all incoming proportions must have been received.

```
"Initialise the state of the fluid";
$i := 0$;
do $i < iterationsCount \rightarrow$
    "Send boundaries to neighbours";
    "Translate non-boundary proportions";
    "Collide non-boundary proportions";
    "Copy received boundary data"; /*blocking operation*/
    "Collide boundaries";
    $i := i + 1$
od
```

Table 2. LB task algorithm

The algorithm showed in Table 2 is run by the tasks of a LB job. It highlights the modifications presented previously. The "*Copy received boundary data*" is a blocking operation which causes all tasks to be almost synchronized in their current iteration i.

2 Load balancing with LB tasks

The LB tasks have to be moldable, i.e. they must be able to handle any number of sub-lattices and be dynamically re-configurable (receive new sub-lattices, remove others). Actually, all tasks are handled by a centralized controller which configures them. Obviously, tasks must first register themselves with the controller. Load balancing is simply achieved with "smart" tasks configurations, i.e. how to distribute m sub-lattices among tasks. The present method is general as it can be adapted to any kind of lattice type (D2Q9, D3Q27. . .). Moreover, it doesn't require any adaptation of the base algorithm.

2.1 Homogeneous tasks

If no balancing is done, each task receives the same amount of sub-lattices. This leads to the situation where resources with fast CPUs wait for resources with slow CPUs. This is a consequence of the time synchronization of the tasks (see Section 1.3).

A first step towards load balancing would be to take into account the amount of CPUs available to the tasks. Each task receives a number of sub-lattices proportional to its number of available CPUs. This configuration leads to what we call "homogeneous tasks".

2.2 Scaled tasks

The previous method could be enhanced by taking into account the speed of the CPUs. Indeed, with homogeneous tasks, each CPU receives the same amount of work. The fast CPUs will wait for the slower ones at each LB iteration. It results that the overall LB process is limited by the slowest CPU. Therefore, "more work" should be attributed to a fast CPU. For example, a CPU i twice as fast a CPU j should receive twice as much work as CPU j. This configuration leads to what we call "scaled tasks".

A weight w_i can be associated to each task i. It is calculated in the following way:

$$w_i = \frac{c_i p_i}{\sum_i c_i p_i} \tag{3}$$

where c_i is the number of available CPUs and p_i their power. If m is the number of sub-lattices to distribute among tasks, each task i receives $\lfloor w_i m \rfloor$ sub-lattices. It therefore remains $k = m - \sum_{i=1}^{n} \lfloor w_i m \rfloor$ sub-lattices to distribute. They are distributed to the most "underestimated" tasks, i.e. those that maximize $w_i m - \lfloor w_i m \rfloor$.

The "scaled tasks" configuration requires to know the p_i values. These are estimated by using a benchmark. It is a small classical non-parallel LB algorithm with s LB sites. The LB algorithm is iterated l times. If it took t_i seconds to execute the benchmark, $p_i = \frac{s \times l}{t_i}$. The power p_i is given as a number of sites processed per second.

3 Estimation of execution time

In this section, a model to estimate the execution time of a LB job is presented. It will allow to estimate theoretically an upper bound on the gain that can be obtained with the presented load balancing technique.

The *normalized execution time* t of a LB job is the time (in seconds) needed by all LB tasks to execute one LB iteration. The normalized execution time t_i of a task i is the time (in seconds) the task needs to execute one LB iteration. As all tasks are synchronized, all tasks have done one iteration after $t = \max_i t_i$ seconds.

t_i is given by $t_i^c + t_i^t$ where t_i^c is the processing time and t_i^t the transmission time (when exchanging inter-tasks data for one iteration).

The processing time t_i^c depends on the number of CPUs c_i available to task i, their power p_i (we consider that all CPUs of one resource have the same power) and q_i the amount of work assigned to task i. It is calculated with the following expression:

$$t_i^c = \frac{q_i}{c_i p_i}$$

where q_i is the number of LB sites that task i handles. If p_i is given as the number of LB sites processed per seconds, t_i^c is given in seconds.

The transmission time t_i^t depends on d_i the amount of data exchanged each iteration and BW the network bandwidth. It is calculated with the following expression:

$$t_i^t = \frac{d_i}{BW}$$

where d_i is the amount of data (in LB sites) exchanged each iteration. If BW is given as a number of LB sites transmitted per second, t_i^t is given in seconds.

The d_i parameter depends on the sub-lattices assigned to task i. Sub-lattices of a given task that share interfaces (faces or edges) exchange the data associated to these interfaces through memory, not through the network. In the best case, the amount of data exchanged through network is minimized by a good placement of sub-lattices ($d_i = \hat{d}_i^{min}$). In the worst case, no data are exchanged through memory ($d_i = d_i^{max}$). The $\hat{d}_i^{min}$ case is not simple to estimate. An approximation d_i^{min} is therefore made. All the sites of the sub-lattices of a given task are artificially grouped into a single cubic sub-lattice. The cube minimizes the surface of the faces of a cuboid given a fixed volume. The amount of data exchanged by the sub-lattice, proportional to the surface of its faces, is therefore minimum. Actually, $d_i^{min} \leq \hat{d}_i^{min}$ because sub-lattices cannot always be organized to form a cube.

The BW parameter depends on the bandwidth b of the network (given as a number of bits transmitted per second) and the size of a LB site. Our model is a D3Q19 so a site is "represented" by 19 floats. If a float is composed of 4 bytes, a site "weighs" 608 bits. BW is calculated with the following expression:

$$BW = \frac{b}{size(site)}$$

4 LaBoGrid

LaBoGrid is a modelling platform based on LB methods and adapted to Grid computing. It is written in Java, so it can be deployed on any computer provided that a Java virtual machine is available. LaBoGrid's implementation is based on the principles described so far.

LaBoGrid is based on an "in house" middleware providing a hierarchy of classes and interfaces that can be extended and implemented to gridify the LB modelling. The two main components of this middleware are the controller and the distributed agent. The distributed agents run on grid resources. The controller connects them together. A controller task can be associated to the controller. This special task instantiates the tasks of a job on the distributed agents. Figure 4 illustrates the LaBoGrid components and how they are connected.

There are two classes that need to be extended to adapt the middleware to a given context: the controller task (LBCT) and the distributed agent task (LBDAT).

The LBCT instantiates a LBDAT on each resource. It then distributes the sub-lattices among the tasks.

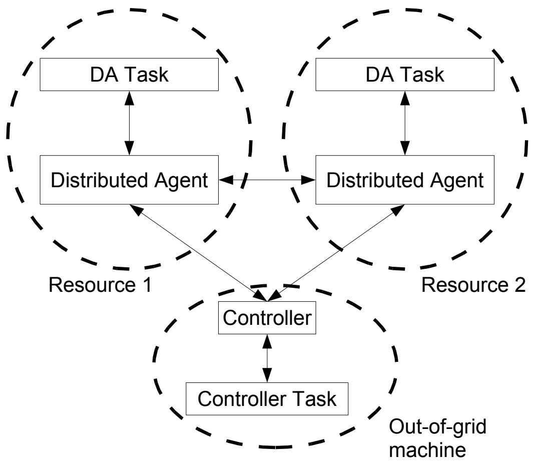

Fig. 4. LaBoGrid components and their interconnections

Given its configuration, the LBCT uses either “scaled tasks” or “homogeneous tasks” (both introduced in section 2). With “scaled tasks”, the LBCT must instantiate a benchmark on each registered task before the instantiation of the LB tasks (see Section 2.2).

The LBDAT runs the algorithm showed in Table 2.

5 Results

In this section, the estimated normalized execution time obtained with homogeneous tasks and scaled tasks is compared. Estimated and observed job execution times then will be compared.

A LB modelling using an initial lattice of $176 \times 176 \times 176$ LB sites divided into 125 sub-lattices is run on a small grid of 10 machines. There are 4 Celeron based PCs (1 CPU each), 4 Pentium IV based PCs (1 CPU each) and 2 Xeon based servers (4 cores each). All theses machines are connected through a 100Mbits/s Fast ethernet network.

Figure 5 shows the estimated normalized execution times on each machine. In the case of homogeneous tasks, the difference between execution time on Celeron based machines and the servers is large. This is because the Xeon processors of the servers are the fastest of the grid and the Celeron the slowest.

With scaled tasks, this difference is much smaller when $d_i = d_i^{min}$. When $d_i = d_i^{max}$, there is again a large difference between execution times on both Celeron machines and Xeon machines. But in this case, the Xeon machines are the slowest. This is because, as they have more sub-lattices to process than Celeron machines, they have more data to exchange. Their transmission time is

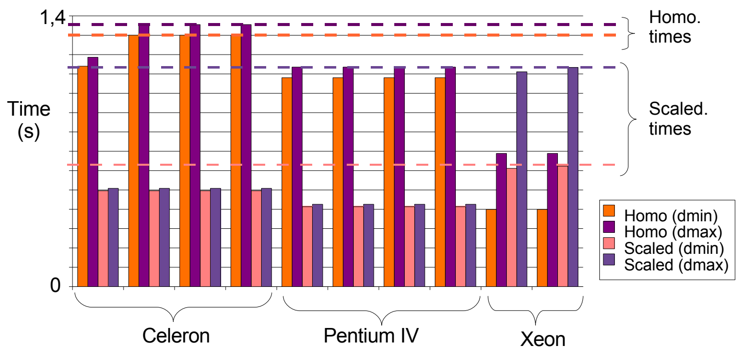

Fig. 5. Estimated normalized task execution times

therefore increased. In the $d_i = d_i^{min}$ case, the transmission time difference is rather small. This is not the case when $d_i = d_i^{max}$.

The are also small differences in processing times caused by the quantification error introduced by the sub-lattices.

The estimated normalized job execution times are represented by the horizontal dashed lines on Figure 5. On the experimental grid, a job executed with scaled tasks can run up to twice as fast as a job with homogeneous tasks. However, this is in the case where $d_i = d_i^{min}$. With $d_i = d_i^{max}$ there is still some gain but it is smaller.

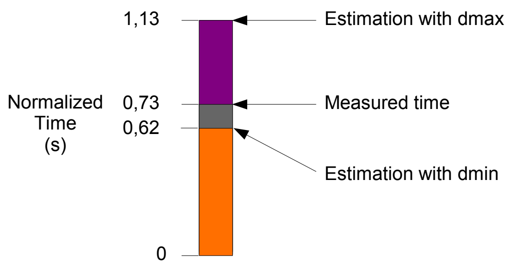

Fig. 6. Comparison between estimated and observed job execution times

Actually, sub-lattices should be placed carefully among the tasks to minimize the network communications. In LaBoGrid, this is done with a simple partitioning algorithm based on some heuristics (with a complexity of $O(n^2)$ in the number of sub-lattices). The comparison between estimated and observed job execution time given in Figure 6 shows it works rather well. Indeed, the observed execution time is near the $d_i = d_i^{min}$ estimated case.

6 Conclusion

After a theoretical presentation of Grid computing, LB methods and how they can be adapted to the Grid environment, a method to handle CPUs heterogeneity among the machines of a Grid has been presented. Indeed, all these machines don't necessarily have the same number of CPUs and CPU speeds. Using this load balancing method led to rather good results. A model to estimate the execution time of LB jobs has been given. Our implementation of a distributed 3D LB modelling system adapted to the Grid environment, LaBoGrid, has been outlined.

This is a first step toward Grid enabling LaBoGrid. In the future, the scalability of such a system should be analysed. Indeed, we work in a rather good context, i.e. a few machines connected by a 100Mbits/s network. Another important step towards an adaptation to the Grid environment is fault tolerance: presently, if a machine interrupts its work and goes down, the overall system is stopped and can't continue. Even worse, the data of the down machine are lost: the global data cannot therefore be reconstructed. Distributed checkpointing is a possible solution. Each task could save its state (actually, the data it processes) to some other resources. When a resource is lost, the data of its task can be retrieved and distributed to other tasks.

Acknowledgements

This work was performed in the frame of a research concerted action financed by the Communauté Française de Belgique.

References

1. J. Nabrzyski and J. Schopf and J. Weglarz: Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers (2003)
2. I. Foster: What is the Grid? A three point checklist. Grid Today (2002)
3. A. Dupuis: From a Lattice Boltzmann model to a parallel and reusable implementation of a virtual river. PhD Thesis, Université de Genève. (2002)
4. S. Wolfram: Cellular automaton fluids 1: Basic theory. J. Stat. Phys. **45** (1986) 471–526
5. Q. Zou and X. He: On pressure and velocity boundary conditions for the Lattice Boltzmann BGK model. **9** (1997) 1591–1598
6. Y.H. Qian and D. d'Humières and P. Lallemand: Lattice BGK models for Navier-Stokes equation. Europhys. lett., **17(6)** (1992) 470-484