Large Scale Lattice-Boltzmann Simulations: Computational Methods and Applications

Kandhai, B.D.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 5
Load Balancing in Lattice-Boltzmann Simulations

5.1 Introduction

Realistic lattice-Boltzmann simulations often require large amounts of computational resources and are therefore executed on parallel systems. Generally, parallelization * of Lattice-Boltzmann simulation programs is based on one- and two-dimensional decomposition of the computational grid in equal sub-volumes, and load balancing is completely ignored for simplicity. These decompositions are therefore only efficient when the workload is distributed homogeneously † over the lattice.

However, in many fluid dynamical problems, the workload may be highly non-homogeneous and sometimes can even vary dynamically during the simulation. To facilitate efficient simulations of these problems, we developed a generic, portable parallel lattice-Boltzmann simulation program which can handle three domain decomposition strategies, namely slice and box decomposition and the Orthogonal Recursive Bisection (ORB) method. The ORB method can be used to generate approximately balanced decompositions by taking into account the workload on each lattice point. In this thesis we will restrict ourselves to applications with a static workload distribution.

To illustrate the usefulness of the different load balancing strategies we applied the methods to two realistic test cases. The first one is fluid flow through a massive random fibre network which poses a realistic model for man-made fibrous webs such as paper [57]. In this model, flexible fibres are positioned randomly over the grid on a flat substrate. Slice and box decomposition are therefore ex-

*This chapter is based on the following publication:


†By homogeneous workload distribution we mean that obstacles in the fluid are distributed uniformly over the lattice, while in a heterogeneous workload distribution this is not the case.
54 Load Balancing in Lattice-Boltzmann Simulations

ected to be sufficient to achieve a relatively balanced and efficient parallel simulation. To clarify the idea of load balancing through the ORB technique, we also simulated fluid flow in a tube of varying cross-section, an example taken from a medical setup where blood cells are isolated due to their differences in sedimentation velocity [58]. It is shown that high parallel efficiencies can be obtained for both homogeneously and heterogeneously distributed workloads, thus supporting efficient simulations of a variety of realistic systems.

We shall first demonstrate that parallel computing is a practical technique to fulfill the computational requirements for simulating fluid flow in representative random geometries. Then we shall discuss in detail the different parallelization strategies, and study both homogeneously and heterogeneously distributed workloads.

5.2 Parallel computing in complex flow simulations

Parallel computing is often a prerequisite for achieving appropriate computing times and for fulfilling the memory requirements of large-scale 3D simulations. In this section we will shortly discuss the average computational requirements for simulating fluid flow in a prototypical realistic simulation, namely fluid flow in random fibrous media. The main emphasis is to illustrate the role of parallel computation in simulating realistic applications by the lattice-Boltzmann method.

As we have shown in chapter 3, the main artifacts of lattice-Boltzmann simulations are standard discretization errors and boundary effects due to the bounce-back rule. Apart from the error sources of the lattice-Boltzmann method, the computational requirements are also determined by the physical dimensions of the application to be simulated. For example, in the case of flow in representative random fiber mats, a square sample with an edge length of 4 fibre lengths in the horizontal plane and a height of 10 fibre thicknesses has to be used (details can be found in chapter 7. Moreover, due to the error sources of LBM, we found that lattices of 400x400x60 or 800x800x110 points (where an additional fluid layer of thickness 10 is included) are required to simulate a representative network with an accuracy of 15% (see chapter 9 for details). To represent the larger lattices using 32-bit real numbers, a memory capacity of the order of 5.4 Gb is required.

Moreover, estimates of timing requirements show that the real simulations would require approximately three hours of computing time on the full domain of a 64-node Cray T3E system (with 128 Mb of main memory for each Digital Alpha EV5 300 Mhz processor) [59]. On a single processor it would take about 200 hours of computing time (assuming parallel efficiencies close to one). This is a serious bottleneck especially since in the present problem it is necessary to simulate several samples in order to find statistically confident correlations between permeability and various structural parameters of the fibre network.
5.3 Load balancing in a homogeneous workload distribution

It is therefore clear that parallel computing is the only possibility to fulfill the large computational requirements of the present prototype simulation and presumably of many other related practical problems.

5.3 Load balancing in a homogeneous workload distribution

Basically parallelization of grid based algorithms like those of finite-difference, finite-element and lattice-Boltzmann models is done by means of the data decomposition strategy where the computational grid is decomposed into sub-domains [60]. Each processor performs computations on a certain sub-domain and exchanges information with other nodes in order to resolve dependencies. The two factors controlling the efficiency of parallelization are the ratio between the communication time and the computation time, and the balance of workload among the processors. In this section we will study parallelization of problems with a homogeneously distributed workload such as fluid flow in the random fibre network.

5.3.1 Slice and box decomposition

For problems with a homogeneous workload, parallelization can be done by means of a straightforward decomposition of the computational grid in equal sub-volumes, as discussed in Refs. [61, 62, 63]. With a moderate number of nodes, the decomposition may be done in one dimension ('slice decomposition') or in two dimensions ('box decomposition'). These methods depend only on the dimensions of the lattice and on the number of processors while the geometry of the application itself is neglected. An important feature of the lattice-Boltzmann schemes in this context is the inherent spatial locality of the collision operator. Furthermore, interactions between processors are only required at the propagation step. At this step particles on a border node can move to a lattice point in the domain of a neighboring processor or vice versa. By using a ghost layer of lattice points in the surroundings of the sub-domain, the propagation step can be isolated from the data exchange step. After the propagation step, the values in the ghost layer are send to the neighboring processors (communication process). The advantage of the slice and box decomposition is that the dependencies between the processors are simple due to the regular connectivity of the lattice and the fact that the partitions are of equal size.

The simulation program that is used in the present fluid flow problem was implemented in the programming language C. Parallelization was accomplished using features of the MPI message passing library [64].
5.3.2 Performance model and results

To get insight in the performance of slice and box decomposition, we derive first a simple time-complexity model with the following assumptions.

1. No overlapping in computation or communication between nodes.

2. No load imbalance due to the geometry of the problem. The only source of load imbalance may be caused by the number of lattice points being not divisible by the number of processors.

The turnaround time of one time iteration, $T(N, p)$, is the sum of the computation time, $T_{\text{calc}}(N, p)$, and the communication time, $T_{\text{comm}}(N, p)$, where $N = (N_x, N_y, N_z)$ gives the number of lattice points in each direction and $p$ is the number of processors. For the slice decomposition (in the $x$ direction),

$$T_{\text{calc}}(N, p) \sim \frac{N_x}{p} |N_N_z| \tau_{\text{calc}}, \quad \text{and} \quad T_{\text{comm}}(N, p) \sim 20N_yN_z \tau_{\text{comm}},$$

where the factor 20 originates from the number of neighbors (left and right), the number of communication actions per neighbor (send/receive), and the number of lattice vectors to be exchanged. The parameters $\tau_{\text{calc}}$ and $\tau_{\text{comm}}$ represent the time needed to perform one lattice-Boltzmann grid point update and the time needed to send one link value from a processor to a neighboring processor, respectively. In this model we neglected the communication latency. Analogously, it can be shown that, for the box decomposition (for an equal number of partitions in the $x$ and $y$ directions), the computation and communication times can be approximated by

$$T_{\text{calc}}(N, p) \sim \frac{N_x}{\sqrt{p}} |N_N_y N_z \tau_{\text{calc}}, \quad \text{and} \quad T_{\text{comm}}(N, p) \sim 20\left(\frac{N_x}{\sqrt{p}} + \frac{N_y}{\sqrt{p}}\right)N_z \tau_{\text{comm}}.$$

The relative efficiency $\varepsilon$, a measure for the scalability of a parallel program, is defined as the ratio $\frac{P_0 T(N, p)}{P T(N, p)}$ [62, 60], where $P_0$ is the minimum number of processors for which the job has been run. For these decompositions the relative efficiencies can be approximated by

$$\varepsilon_{\text{slice}} \sim \frac{1}{1 + \frac{20p}{N_x} \frac{\tau_{\text{comm}}}{\tau_{\text{calc}}}} \quad \text{and} \quad \varepsilon_{\text{box}} \sim \frac{1}{1 + 20\left(\frac{\sqrt{p}}{N_x} + \frac{\sqrt{p}}{N_y}\right) \frac{\tau_{\text{comm}}}{\tau_{\text{calc}}}}. \quad (5.1)$$

In Figs. 5.1 and 5.2 we plotted the measured relative efficiencies for slice and box decomposition together with a fit to the corresponding performance model for fibre networks of dimensions 100x100x60 and 200x200x110, respectively. We emphasize that the purpose of these fits is to gain more insight into the scalability of these decompositions with respect to the number of processors and the lattice dimensions. An actual performance prediction based on machine parameters is not very useful here due to the simplicity of the model. The computations were performed on a 64-node Cray T3E system. The fits by the theoretical model, Eqs. (5.1), on the computed results for 100x100x60 and 200x200x110
5.3 Load balancing in a homogeneous workload distribution

Figure 5.1: The relative efficiency as a function of the number of processors. The test problem is flow in random fibre networks on a lattice of dimensions 100x100x60. Solid line (with marked points) is the result for slice decomposition and dashed line (with marked points) is that for box decomposition. A least squares fit by the theoretical model is shown for both decomposition strategies.

lattice give for the slice decomposition \( \frac{t_{\text{comm}}}{t_{\text{calc}}} = 0.06 \) and 0.07, respectively and for the box decomposition \( \frac{t_{\text{comm}}}{t_{\text{calc}}} = 0.07 \) and 0.08, respectively.

The fluctuations of the data by almost 15% is due to the effect of caching and the load imbalance (when the number of lattice points is not evenly divisible by the number of processors). Similar fluctuations are also seen when flow in a geometry with no obstacles at all, is simulated (data not shown). It is evident from these experiments that slice and box decompositions are both efficient and useful strategies for problems with a homogeneous workload distribution. Slice decomposition is relatively easy to implement, but its efficiency is satisfactory only when the number of processors is relatively small (below 20, say). For the box decomposition, the efficiency with a lattice of 100x100x60 is around 0.9 on 64 processors.

For a large number of processors, it may become useful to extend these strategies to three dimensional cubic decompositions for which the relative efficiency
Figure 5.2: The relative efficiency as a function of the number of processors. The test problem is flow in random fibre networks on a lattice of dimensions 200x200x110. Solid line (with marked points) is the result for slice decomposition and dashed line (with marked points) is that for box decomposition. A least squares fit by the theoretical model is shown for both decomposition strategies.

The relative efficiency can be approximated by,

\[ \varepsilon_{cubic} \sim \frac{1}{1 + 20 \left( \frac{p_x^3}{N_x} + \frac{p_y^3}{N_y} + \frac{p_z^3}{N_z} \right) \frac{\tau_{comm}}{\tau_{calc}}} \]

5.4 Load balancing in a heterogeneous workload distribution

In the previous section we assumed that load imbalance among different processors is negligible. This is true for a random fibre network because the fibres are more or less homogeneously distributed in space. For a heterogeneously distributed workload, slice and box decompositions can result in a considerable load imbalance. Consequently the efficiency of the parallel program will decrease due to idle synchronization times. Also, one of the main advantages
of the lattice-Boltzmann method is its suitability for a large class of different geometries, and in many fluid-dynamical problems workload is certainly not distributed homogeneously over the grid due to the geometry of the problem. It is therefore important to study decompositions that can deal with heterogeneously distributed workloads. The need for such load balancing approaches was already noticed in Ref. [61]. In this section we will analyze these issues in detail. For a typical example we take the simulation of flow in so-called centrifugal elutriation chambers [58, 65, 66, 67]. In centrifugal elutriation, human blood cells are isolated by means of differences in their sedimentation coefficients. The core of this technique is the combined effect of centrifugal and hydrodynamic forces acting on a blood cell moving in a rotating chamber. The geometry of the chamber (see Fig. 5.3) is a critical factor in the separation process [58], and is such that straightforward slice and box decompositions will create significant load imbalance among the processors.

5.4.1 Domain decomposition

The first step in load balancing is to find a partitioning of the grid such that differences in the workload of the processors are minimized. There are several ways to accomplish this goal, namely

1. Orthogonal Recursive Bisection: the computational grid is decomposed into partitions in an orthogonal direction, such that the workload is balanced. On each partition the same procedure is applied recursively. Workload due to obstacles in the fluid is neglected. The ORB method will create decompositions like the one in Fig. 5.3.

2. the Recursive Spectral Bisection method [60]: in the Recursive Spectral Bisection method both the connectivity of the grid and the workload at each lattice point are taken into account to find partitions with an optimal balance of both communication and computation;

3. load balancing by means of optimization strategies [68]: in this strategy, the computation and communication times are modeled by means of a cost function, and a partitioning which minimizes this cost function is approximated by means of optimization methods like the Simulated Annealing or Genetic Algorithms.

We have chosen the ORB method for two reasons. First of all, from a practical point of view minimization of the computation time is more important than minimization of the communication time. It became evident in our previous timing experiments that in the lattice-Boltzmann method the communication overhead is small for moderate lattice sizes. Secondly, the other two strategies are quite expensive and should be used when the communication time becomes significant.
5.4.2 Processor dependencies and performance results

The major difference between the ORB partitioning and the slice and box decompositions is the communication pattern of the processors. Each processor can now have a varying number of neighbors, and the interprocessor dependencies are more complicated for the corner points. In Fig. 5.4 this irregular communication pattern is clearly illustrated for the decomposition of Fig. 5.3. In the west direction e.g., processor 5 will now exchange data with processors 1, 2 and 3, while processor 4 will only communicate with processor 1. Also the amount of data to be exchanged between the neighboring processors is not the same for each neighbor. This irregular communication pattern depends completely on the grid partitioning and thus on the geometry of the problem.

![Figure 5.3: The elutriation-chamber geometry (tube of varying cross-section) and a 3x3 partitioning based on the orthogonal recursive-bisection method. Dark and white regions are solid and fluid space respectively.](image)

![Figure 5.4: The irregular interprocessor dependency (for the decomposition of Fig. 5.3) in case of processors 3 (dashed), 4 (dotted) and 5 (solid arrows). The inter-processor dependencies are different for each node and depend on the decomposition.](image)

We implemented the ORB strategy and performed timing measurements for the test problem of Fig. 5.3. The calculation time per processor is shown in Fig. 5.5 for a run on 16 nodes.
Figure 5.5: Execution profiles for the different decomposition methods. In this figure we clearly see that the slice and box decompositions are inefficient for this application. The execution profile obtained for the ORB decomposition is approximately balanced.

In this picture we included the cases where slice and box decomposition are applied to the same problem. It is evident that indeed both slice and box decomposition are inefficient approaches for this application. The workload between the processors is approximately balanced when the ORB method is used, while in slice and box decomposition we see big differences in the calculation times of the different processors.

<table>
<thead>
<tr>
<th>p</th>
<th>$T_{\text{Slice}}$</th>
<th>$T_{\text{Box}}$</th>
<th>$T_{\text{ORB}}$</th>
<th>$\frac{\min(T_{\text{Slice}}, T_{\text{Box}})}{T_{\text{ORB}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53.3</td>
<td>53.5</td>
<td>53.7</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>33.2</td>
<td>33.1</td>
<td>27.8</td>
<td>19.1 %</td>
</tr>
<tr>
<td>4</td>
<td>18.2</td>
<td>15.7</td>
<td>14.0</td>
<td>12.1 %</td>
</tr>
<tr>
<td>9</td>
<td>8.7</td>
<td>9.1</td>
<td>7.1</td>
<td>22.5 %</td>
</tr>
<tr>
<td>16</td>
<td>5.2</td>
<td>5.3</td>
<td>4.1</td>
<td>26.8 %</td>
</tr>
</tbody>
</table>

Table 5.1: The average execution time for 5 time iterations obtained by the different decomposition strategies for the test problem of Fig. 5.3. The problem size was 150x50x100 points. In the last column we show the benefit, which is defined relative to the minimum of the turn around times of slice and box decompositions. It is evident that for this specific test case slice and box decomposition are 12 to 27% less efficient compared to ORB.
Table 5.2: The average execution time for 5 time iterations obtained by the different decomposition strategies for the test problem of Fig. 5.3. The problem size was 139x380x50 points. In the last column we show the benefit, which is defined relative to the minimum of the turn around times of slice and box decompositions. It is evident that for this specific test case slice and box decomposition are 18 to 60 % less efficient compared to ORB.

<table>
<thead>
<tr>
<th>p</th>
<th>Tslice</th>
<th>Tbox</th>
<th>TORB</th>
<th>min(Tslice, Tbox)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>57</td>
<td>44.9</td>
<td>38</td>
<td>18.2 %</td>
</tr>
<tr>
<td>9</td>
<td>31.1</td>
<td>31.3</td>
<td>19.5</td>
<td>59.5 %</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>22.7</td>
<td>16.8</td>
<td>35.2 %</td>
</tr>
<tr>
<td>16</td>
<td>18.8</td>
<td>17.6</td>
<td>13.3</td>
<td>32.3 %</td>
</tr>
<tr>
<td>20</td>
<td>15.5</td>
<td>14.9</td>
<td>11.6</td>
<td>28.5 %</td>
</tr>
<tr>
<td>24</td>
<td>13.5</td>
<td>11.9</td>
<td>7.6</td>
<td>56.6 %</td>
</tr>
</tbody>
</table>

The results of timing measurements for different number of processors are shown in table 5.1 and 5.2. The ORB method is on the average 12 to 27% and 18 to 60% more efficient than the slice and box decomposition for a lattice of 150x50x100 and 139x380x50 points respectively. We expect that similar timing behavior will be found when both the problem size and the number of processors are increased. The benefit gained by load balancing will certainly be significant for large-scale simulations (execution time of many hours) and even higher for complicated fluid-dynamical problems like simulation of particle-suspensions in a chamber. The important point we want to emphasize here is that the extra communication overhead due to the irregular communication pattern is small. Extension of these ideas to dynamically varying workloads (as present in simulation of growth phenomena by convection-diffusion [6]) looks promising, especially because of this relatively small overhead.

Another important point is parallelization of lattice-Boltzmann models based on non-uniform lattices. It has been argued that one of the serious shortcomings of the lattice-Boltzmann method is the uniform nature of the lattice. Lattice-Boltzmann models based on non-uniform lattices have been proposed recently by several authors (see for example Refs. [69, 29] and addendum A). Parallelization of these models will certainly profit from using this load balancing approach.

5.5 Conclusions

We discussed a generic and efficient parallelization of the lattice-BGK method. The most straightforward parallelization is based on slice and box decomposition of the computational grid. By using this approach we obtained efficiencies varying between 0.8 and 0.9 for lattices of 100x100x60 points on 64 processors of a Cray T3E system. This approach is limited to problems with a homogeneous
workload distribution. A property of the lattice-Boltzmann method is its suitability to a wide range of geometries. It is therefore important to study load balancing issues related to this method. We applied the ORB method to generate decompositions with approximately equal workloads. The main difficulty is that the communication pattern between the processors now becomes irregular. We found that the extra communication overhead associated with this irregular communication pattern is small and, for a test problem, on the average 12 to 60% increase in speed was gained in comparison with the slice and box decomposition.

We restricted our study to static workloads since all the applications that will be discussed in this thesis have a static computational geometry. However, in our research group much effort is put towards the dynamic load balancing subject. Recently, Schoneveld et al [70] studied rigorously dynamic load balancing strategies for Lattice-Boltzmann simulations with varying computational workloads. For this a generic framework based on dynamic data structures was developed that allows migration of computational cells among processors. As a test case a simulation of coral growth was considered. In this test case the real biophysical problem is to investigate the effect of flow and diffusion on the morphology of a model of a stony coral [6]. It was demonstrated that appropriate dynamic load balancing may have significant effect on the parallel performance of the simulations. Overeinder et al [71] extended the PVM parallel programming library with features for automatic balancing of computational tasks (Dynamic PVM library). In fact, they monitored the workload on each processor during a simulation. Whenever a significant imbalance in the computational work between the different processors is observed some of the tasks executed on a machine with a high load are migrated to an idle or less occupied machine. For different applications an average speedup of around 20% was gained.

Our aim is to extend our lattice-Boltzmann simulation kernel with functionalities to deal with multi-phase flows and moving particles and to combine that with the ideas for dynamic load balancing proposed by Schoneveld et al [70] and Overeinder et al [71]. In this way a lattice-Boltzmann simulation environment which is generic from both a parallel-computing and an application point of view will be realized. We believe that such an approach, which is typical in the field of computational science, is a fruitful strategy for modeling and simulation of a large class of realistic fluid-dynamical problems of great complexity.
64

Load Balancing in Lattice-Boltzmann Simulations

5.5 Conclusions

We discussed a generic and efficient parallelization of the lattice-Boltzmann method. The most straightforward parallelization is based on site and box decomposition of the computational grid. By using this approach we obtained efficiencies ranging between 0.8 and 0.9 for test cases of 100^3 to 1024^3 points on 64 processors of a Cray T3E system. This approach is limited to problems with a homogeneous