Parallelization strategies and efficiency of CFD computations in complex geometries using Lattice Boltzmann methods on high-performance computers

M. Schulz, M. Krafczyk, J. Tölke and E. Rank

Lehrstuhl für Bauinformatik, Technische Universität München, 80290 München, Germany

Abstract. A frequently stated property of the Lattice Boltzmann (LB) method is, that it is easy to implement and that the generation of computational grids is trivial even for three-dimensional problems. This is mainly due to the usually chosen approach of using full matrices to store the primary variables of the scheme. However this kind of implementation has severe disadvantages for simulations, where the volume of the bounding box of the flow domain is large compared to the actual volume of the flow domain. Thus the authors developed data structures which allow to discretize only the fluid volume including boundary conditions to minimize memory requirements, while retaining the excellent performance with respect to vectorization of standard LB-implementations on supercomputers. Due to extensive communication hiding using asynchronous non-blocking message transfer an almost linear parallel speedup is achieved.

1 Introduction

In the past years, the Lattice-Boltzmann method [1,2] was developed as a valuable complementary approach to classical CFD techniques for solving the Navier-Stokes equations. The primary variables of this scheme are normalized particle probability distributions $f_i(t, x), i \in \{0, ..., N-1\}$ which represent the occupation number of particles on lattice nodes with a discrete velocity e_i. The computations in this work are based on the three-dimensional D3Q15 model [1], where $N = 15$.

The evolution dynamics of the f_i is given by the Lattice-Boltzmann equation

$$f_i(x + e_i\Delta t, t + \Delta t) - f_i(x, t) = \Omega_i(x, t) \tag{1}$$

which implies that the computational grid is identical to the lattice defined by the set of vectors $\{e_i\Delta t\}$.

The collision operator

$$\Omega_i = -\frac{1}{\tau}(f_i - f_i^{(eq)}) \tag{2}$$

is based on the Single Time Relaxation Aproximation (STRA) and represents the rate of changes due to particle collisions. The relaxation time

$$\tau = \frac{6\nu + 1}{2} \tag{3}$$

can be used to tune the kinematic viscosity and defines the rate of approach to the equilibrium state which is given by a polynomial of the macroscopic flow properties u and ρ. These can be defined as moments of the f_i:

$$\rho(t, x) = \sum f_i(t, x) \tag{4}$$

$$u(t, x) = \frac{1}{\rho} \sum f_i(t, x) e_i. \tag{5}$$

The dynamics of velocity and pressure (which is related to density by an equation of state, here $p = c_s^2 \rho$, c_s - speed of sound) for small Mach numbers satisfies the Navier-Stokes equations.

Obviously, equation (1) is an explicit Finite-Difference scheme based on a local stencil. For all grid nodes the collision term only depends on the local set of probability distributions of the actual timestep and the computation of the left-hand side of equation (1) requires information from the neighbouring nodes.

A straightforward approach to implement equation (1) is to map the discretized flow domain onto a uniform grid using full matrices to store f_i. The obvious advantages are:

- The generation of grids is trivial, even for complex geometries such as porous media. All nodes are marked with respect to being part of the flow or solid domain by using a flag matrix which can be directly obtained from e.g. computer based tomography.
- Neighbour relationships between nodes are known due to the topological simplicity of the full matrix approach and allow an easy and effective memory access. This in turn permits very effective vectorization and parallelization when using domain decomposition based on geometric slicing of the simulation domain.

Yet for many practical problems there is a serious drawback: If the volume of the flow domain is small compared to the volume of its bounding box, the use of full matrices results in a severe waste of memory.

Concerning parallelization, two additional points have to be taken into account:

- Load balancing:
  Especially when considering inhomogeneous simulation domains, for a huge number of parallel processes it is no longer possible to subdivide the computational domain into slices or cuboids without severe load imbalancing.

- Memory consumption:
  Even if for a moderate number of subdomains resonable load balancing with respect to an equal number of degrees of freedom (DOF) per subdomain can be achieved, the memory consumption per subdomain can vary significantly which may be critical for performance on distributed memory hardware.

To avoid these problems, we developed an efficient data structure which on the one hand maintains most of the advantages of a uniform grid, such as good performance in terms of Flop rate and which on the other hand in many cases drastically reduces memory consumption. In addition, the main problems of using full matrices for weakly connected flow domains in parallel simulations are completely cured.

2 A data structure based on indirect adressing

The propagation step (evaluation of the left hand side of eq. (1)) for an implementation using full matrices for e.g. f_1 assuming $e_1 \equiv (1,0,0)$ typically could be implemented as (pseudo-code):

```
for(x=max_x-1 ; x>0 ; x--)   f1(x,y,z) = f1(x-1,y,z)
```

Using a topologically unstructured grid (i.e. a uniform cartesian grid with "holes"), neighbouring nodes are no longer immediately known. Having lost this natural ordering, one can take advantage to sort the distributions to optimize data flow for the computation of the collision term with respect to the boundary condition type as depicted in Figure 1.

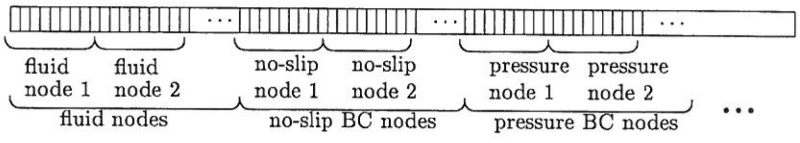

Fig. 1. sorting within the one-dimensional array $f[]$ according to nodal boundary attributes

Propagation now implies copying distributions to arbitrary places within the one-dimensional array. The corresponding origins and destinations have to be stored in precomputed lists. Using only one array for the distributions this information has to be stored using two additional index arrays to store the target and source indices which minimizes memory consumption.

The propagation of e.g. f_1 assuming $e_1 \equiv (1,0,0)$ results in

```
for(i=0 ; i<num_propagations ; i++)  f(target(i)) = f(source(i))
```

where *num_propagations* is the number of propagations required. This approach results in a moderate performance due to the two indirect adressing operations. Using an additional copy array for the f_i only one dereferencing is necessary:

```
for(i=0 ; i<num_propagations ; i++)  f_copy(target(i)) = f(i)
```

This roughly doubles the computational performance at the price of a slight increase in memory requirements.

The following example shows the advantage of using an unstructured grid with respect to memory requirement for a porous flow geometry with a porosity of 20% (Fig. 2, no-slip boundaries are shown in black and surround the flow channels) and the grid resolution of the x-ray tomography of $83 \times 83 \times 30$ was increased using three-dimensional interpolation [5] by a factor of five.

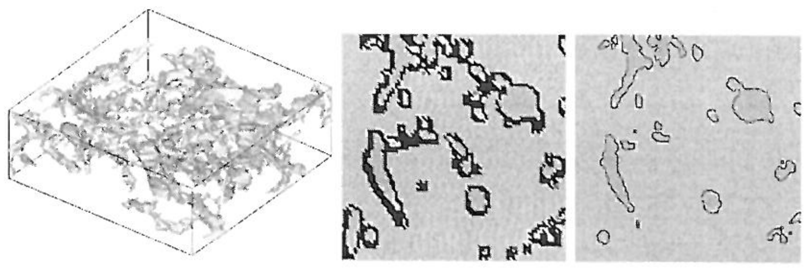

Fig. 2. Porous flow geometry: a) 3D-view, b) slicing for low resolution, c) slicing for high resolution (factor 5)

Depending on the chosen scaling factor (i.e. grid resolution) the expected decrease in memory consumption as compared to the use of full matrices are shown in Fig. 3.

3 Domain decomposition

Having abandoned the use of full matrices the geometric domain decomposition is no longer restricted to produce cuboidal subdomains. Thus we can utilize a state-of-the-art tool based on graph partitioning methods (METIS [3]) to obtain arbitrarily shaped subdomains which leads to very good load balancing even for a large number of subdomains for any flow geometry. An example is given in Fig. 4.

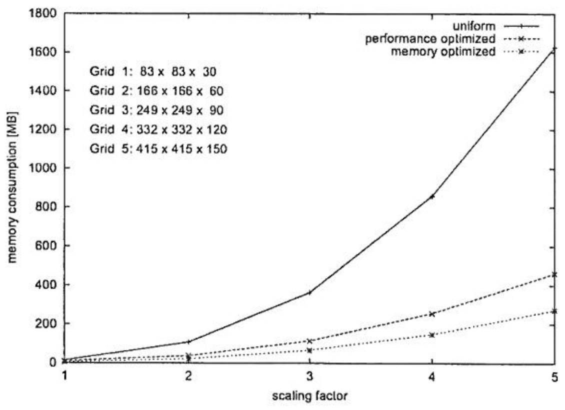

Fig. 3. Memory consumption for different grid resolutions

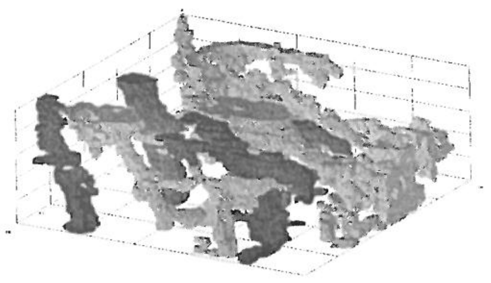

Fig. 4. METIS domain decomposition of a porous medium

4 Parallelization strategy

As indicated above, all necessary information for a nodal update is basically obtained from next neighbouring nodes as is depicted in Figure 5.

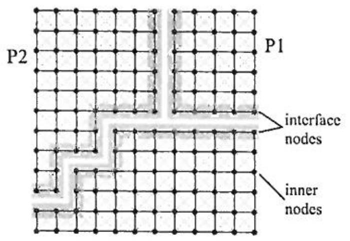

Fig. 5. Cut through the decomposed flow domain indicating the interface width

The information that has to be transferred between pairs of subdomains in each timestep depends on the length and width of the mutual interfaces. Asymptotically, the ratio of the computational effort for the inner and interface nodes is proportional to the inverse of the number of inner nodes, thus a reasonable parallel efficiency can be expected as the cost for the computational overhead due to data exchange becomes vanishingly small for subdomains of increasing size.

There is still room for fine tuning as it is well known that any parallel efficiency can be further improved by partially overlapping the process of data exchange between subdomains and the interface-independant computation (in case of appropriate network hardware). For the LBGK implementation proposed here this overlapping can be optimized by computing the collision term of the interface nodes and a subsequent initiation of their propagation to the corresponding subdomains by using asynchronous non-blocking message passing calls of the underlying MPI library [4]. After this initiation the collision and propagation of the inner nodal distributions (being by far the computationally dominant part) is done. Thus there is additional time left for the completion of the message transfer. Figure 6 shows the difference between the standard approach and the *optimal* one presented here.

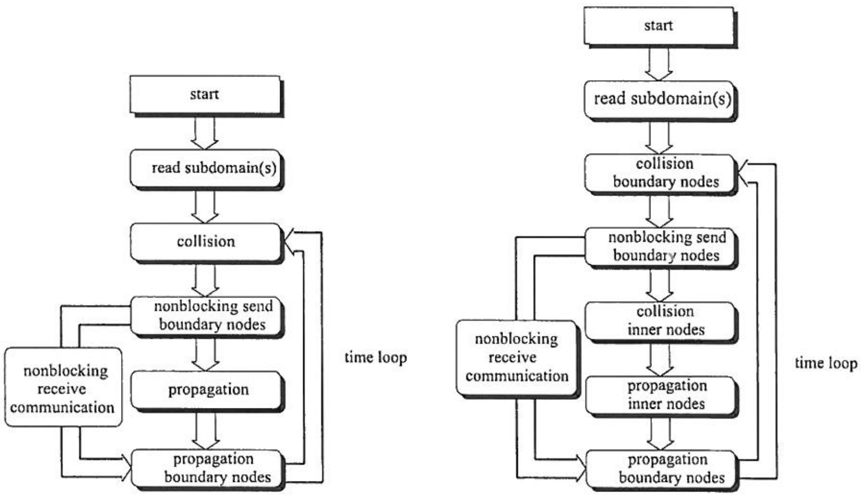
start
read subdomain(s)
collision
nonblocking send boundary nodes
nonblocking receive communication
propagation
propagation boundary nodes
time loop
start
read subdomain(s)
collision boundary nodes
nonblocking send boundary nodes
nonblocking receive communication
collision inner nodes
propagation inner nodes
propagation boundary nodes
time loop

Fig. 6. flow chart for the standard parallel-LB algorithm and the optimal implementation (right)

It should be noted, that the propagation of the interface distributions is naturally taken care of by the communication routine.

5 Results

The data structure explained above was implemented on an eight machine DEC EV 6 workstation cluster as well as on the Bundeshöchstleistungsrechner Hitachi SR8000-F1 being located at the Leibniz-Rechenzentrum Munich. On the Hitachi a two-level parallelization was implemented, as each node of this machine consists of eight processors sharing memory. Thus the auto-parallelizing compiler took care of the intra-node communication. On one computational node we obtain an update rate of up to 2×10^7 (in other words the computation of one timestep of eq. (1) for a grid of 2×10^7 grid nodes takes about one second). This corresponds to about 3 GFlops. The inter-nodal communication was based on the message-passing scheme explained above.

A parallel efficiency of about 0.9 was achieved for various testruns (Fig. 7). Taking into account that the test cases investigated utilized at most 25 % of the nodal memory, the parallel efficiency for subdomains of maximum size can be estimated to be even closer to one.

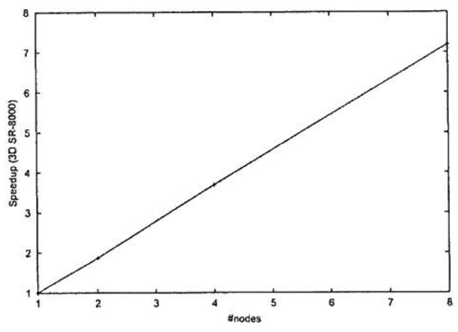

Fig. 7. decomposed flow domain

As a numerical example we computed the friction factor (a non-dimensional measure for the permeability of a flow geometry) for a sphere packing generated by a molecular dynamics type simulation tool [6]. Figure 8 shows one half of the sphere packing inside a cylinder containing more than 1000 spheres. The diameter of one sphere corresponds to thirty grid nodes resulting in an overall system of about 1.7×10^7 nodes. The porosity with respect to the bounding box of the surrounding cylinder is 27 %, the bounding box volume would correspond to 4×10^7 nodes. The computation time (excluding data IO) using four nodes of the Hitachi SR8000-F1 was about 75 minutes.

The resulting friction factor coincides within 3 % with measurements obtained from experiment [7] and direct numerical simulation on a flow geometry with evenly distributed spheres [8].

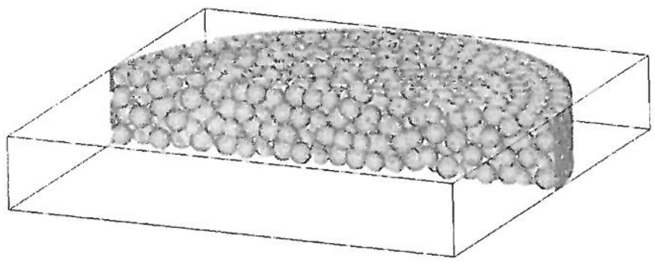

Fig. 8. flow geometry

6 Acknowledgement

The authors would like to thank the HPC group at the Leibniz-Rechenzentrum for valuable technical support. The project is funded by the Deutsche Forschungsgemeinschaft (Ra 624/7-1).

References

1. Quian, Y.H., d'Humieres, D. and Lallemand, P. (1992) Lattice BGK models for N-S equation. Europhys.Lett.**17(6)**, 479–484
2. Krafczyk, M. (2001) Gitter-Boltzmann-Methoden - von der Theorie zur Anwendung. Professorial Thesis LS Bauinformatik TU München
3. Karypis, G., Kumar, V. (1998) Multilevel Algorithms for Multi-Constraint Graph Partitioning.
   http://www-users.cs.umn.edu/~karypis/publications/partitioning.html
4. http://www-unix.mcs.anl.gov/mpi/index.html
5. Engeln-Müllges, G., Reutter, F. (1996) Numerikalgorithmen. VDI Verlag 1996, ISBN 3-18-401539-4
6. Ristow, G. H. (1994) Granular Dynamics: a Review about recent Molecular Dynamics Simulations of Granular Materials. Annual Reviews of Computational Physics **I**, 275–308
7. Durst, F., Haas, R., Interthal, W. (1987) The Nature of Flows through Porous Media. J. Non Newtonian Fluid. Mech **22** , p.169–189
8. Bernsdorf, J., Brenner, G., Durst, F. (2000) Numerical analysis of the pressure drop in porous media flow with lattice Boltzmann (BGK) automata. Comp. Phys. Com.**129**, 247–255