OpenLB: towards an efficient parallel open source library for lattice Boltzmann fluid flow simulations

Vincent Heuveline and Mathias J. Krause

Numerical Methods and High Performance Computing (NUMHPC),
University of Karlsruhe (TH)
Zirkel 2, 76128 Karlsruhe, Germany
{vincent.heuveline,mathias.krause}@rz.uni-karlsruhe.de
http://numhpc.rz.uni-karlsruhe.de

Key words: lattice Boltzmann methods, CFD, high performance computing, parallel computing

1 Introduction

During the last decade lattice Boltzmann methods (LBM) have become a widely accepted tool in fluid dynamics for instance as discrete solvers for the Navier-Stokes equations. The simplicity of the core algorithms as well as the locality properties resulting from the underlying kinetic approach lead to methods which are very attractive in the context of parallel computer and high performance computing. Currently there are intensive research efforts to analyze, develop and adapt adequate lattice Boltzmann approaches for dedicated hardware architectures. Present typical examples of this trend are developments related to IBM Cell processors, Graphic Processing Units, Clearspeed accelerator board and multi-core processors from AMD, Intel and others. These new technologies blur the line of separation between architectures with shared and distributed memory. In that context the authors are convinced that the development of efficient hybrid parallelization schemes for LBM does not only represent a major challenge in the near future but also is a sine qua non condition to take advantage of the performance both on high performance computing (HPC) hardware and on *commodity off the shelf* (COTS) hardware.

In this article we present OpenLB, an open source C++ code for the simulation of 2D and 3D fluid flows by means of LBM. The library is generic and based on advanced software engineering techniques. It offers a platform for LBM developers to produce and exchange code for academic as well as commercial applications. The core of the implementation is a regular, rectangular grid which can be used to build higher-level constructs such as locally refined grids or complex geometries by connecting a set of them together. We focus on a hybrid parallelization concept which allows coping with platforms sharing both the properties of shared and distributed memory systems.

2 Lattice Boltzmann methods (LBM)

A lattice Boltzmann numerical model simulates the dynamics of particle distribution functions $f = f(\boldsymbol{r}, \boldsymbol{v}, t)$ in a phase space with position $\boldsymbol{r}$ and velocity $\boldsymbol{v}$. The continuous transient phase space is replaced by a discrete space with a spacing δr for the positions, a set of q vectors $\boldsymbol{c}_i$ for the velocities and a spacing δt for time. The resulting discrete phase space is called the lattice and is labeled by the term DdQq by the numbers of space dimensions d and the number of discrete velocities q. To reflect the discretization of velocity space, the continuous distribution function f is replaced by a set of q distribution functions f_i, representing an average value of f in the vicinity of the velocity $\boldsymbol{c}_i$. Assuming adequate scaling, the iterative process to be solved in the LB algorithm is written as follows:

$$f_i(\boldsymbol{r} + \boldsymbol{c}_i, t+1) - f_i(\boldsymbol{r}, t) = -\omega \left(f_i(\boldsymbol{r}, t) - f_i^{eq} \left(\rho(\boldsymbol{r}, t), \boldsymbol{u}(\boldsymbol{r}, t) \right) \right) \qquad (1)$$

$$\text{for } i = 0 \cdots q-1,$$

where

$$f_i^{eq}(\rho, \boldsymbol{u}) := \rho t_i \left(1 + 3\, \boldsymbol{c}_i \cdot \boldsymbol{u} + \frac{9}{2} \left| \boldsymbol{c}_i \cdot \boldsymbol{u} \right|^2 - \frac{3}{2} \left| \boldsymbol{u} \right|^2 \right), \qquad (2)$$

$$\rho := \sum_{i=0}^{q-1} f_i \quad \text{and} \quad \rho \boldsymbol{u} := \sum_{i=0}^{q-1} \boldsymbol{c}_i f_i. \qquad (3)$$

The t_i and $\boldsymbol{c}_i$ are lattice dependent constants. It is shown that the LB dynamics is asymptotically equivalent to the dynamics of the Navier-Stokes equations and that the fluid viscosity ν is directly related to the relaxation parameter ω.

3 Hybrid parallelization in OpenLB

3.1 The OpenLB project

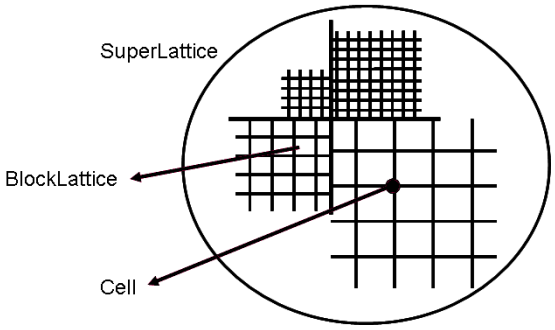
The OpenLB project provides a C++ package for the implementation of lattice Boltzmann simulations that is general enough to address a vast range of problems in computational fluid dynamics. The package is mainly intended as a programming support for researchers and engineers who simulate fluid flows by means of a lattice Boltzmann method. The OpenLB source code is publicly available on the project web site (`http://www.openlb.org`).

3.2 Organization of the code and data structure design

The core of OpenLB consists of a `BlockLattice`, a simple and efficient array-like construct. All `Cells` of the `BlockLattice` are iteratively parsed, and a local collision step is executed, followed by a non-local streaming step. The collision step determines the physics of the model and can be configured by the user, by

attributing a fully configurable dynamics object to each `Cell`. Thus it is easy to implement inhomogeneous fluids which use a different type of physics from one `Cell` to another. Some boundary conditions are non-local and need to access neighbouring nodes. Their execution is taken care of by a postprocessing step which parses selected `Cells` only. Another task that falls under the responsibility of the postprocessing step is the interconnection of various `BlockLattices` which are integrated in higher level constructs called `SuperLattices`. In this way, the efficiency of a basic LBM is combined with an access to advanced constructs, including parallel and grid refined codes.

Fig. 1. Data structure in OpenLB: A number of `BlockLattices` build a `SuperLattice` to adopt higher level software constructs like multi-block, grid refined lattices and parallelized lattices



3.3 Parallel implementation details

The most time demanding steps in LB simulations are the collision and streaming. Since the collision step is purely local and the streaming step only requires data of the neighbouring nodes parallelizing by domain partitioning is efficient because the communication costs are low. It is not categorically necessary to change the basic structure of the sequential lattice Boltzmann algorithm. The usage of pre-processor programming enables to write one single code for the sequential and several parallel modes. In combination with the usage of a modern object oriented and template based programming language, the adoption of several parallelization paradigms like OpenMP and MPI and finally their combination is enabled. To realize a hybrid parallelization in OpenLB, MPI is used for communication between the `BlockLattices` within a `SuperLattice` and OpenMP to parallelize each single `BlockLattice`.

4 Preliminary performance results

In order to test the proposed parallel approaches a D3Q19 lattice of size $201 \times 201 \times 401$ and 100 time steps is considered for the testing. One two-way node with two AMD Opteron sockets running at a clock speed of 2.6 GHz served as a testing environment. Each socket has two cores with 1 MB of level 2 cache. The node is part of the high performance computer *HP XC4000* with 750 nodes, a peak performance of 15.6 TFLOP/s and an InfiniBand 4X DDR Interconnect.

In Figure 2 the efficiency of a purely MPI based approach is presented and compared to the corresponding results of the OpenMP versions. Remark that the considered implementation tested on one node uses a direct access to the local shared memory. It is important to notice that for the case of three and four processes the version executed on three respectively four different nodes i.e. involving internodal communication is more efficient than the version involving intranodal communication. It is observed that the MPI version is more efficient than both OpenMP based approaches, although OpenMP is supposedly dedicated to shared memory environment. The MPI results clearly show the intricate dependency between the efficiency and job partitioning at the core and nodal level.

Fig. 2. Efficiency as a function of the number of processes on *HP XC4000* in comparison with the results of a MPI based parallelization. The abbreviation c/s indicates that the collision and streaming is done in two loops over all cells whereas bulk c/s indicates that only one loop is needed. The asterisked abbreviations mark an approach where the memory is assigned and first touched locally.