

Using OpenMP on a Hydrodynamic Lattice-Boltzmann Code

Gino Bella* Salvatore Filippone†
Nicola Rossi Stefano Ubertini

Università degli Studi di Roma “Tor Vergata”

1 Introduction

The motion of a fluid flow is governed by the Navier-Stokes equations, that, due to their differential and non-linear nature, except in particular cases, are numerically solved using well known techniques such as finite elements and finite volumes methods. Therefore the computational fluid dynamics (CFD), in its conventional meaning, computes pertinent flow fields in terms of velocity \vec{u} , density ρ , pressure P and temperature T , by solving the Navier-Stokes equations in time t and space \vec{x} .

Recently the Lattice Boltzmann Method (LBM) has become an alternative approach to solve fluid dynamics problems, without employing Navier-Stokes equations. This method, introduced at the turn of the 1980s, was directly developed from its predecessor, the lattice gas cellular automata (1964) [5, 7], and, due to the refinements and the extensions of the last years, it has been used to successfully compute a number of nontrivial fluid dynamics problems, from incompressible turbulence to multiphase flow and bubble flow simulations [1, 11, 6, 9, 13]. However, when complex geometries must be solved, LBM still lags behind state-of-the art CFD techniques, due to the inability of LBM to treat non-uniform spatial distribution of mesh grid points.

A computational advantage of the LBM method is that its implementation leads to computational kernels that are very suitable for parallelization in both shared and distributed memory paradigms; we discuss the results we have obtained by taking a straightforward serial implementation and parallelizing it onto a shared memory multiprocessor by using the OpenMP directives [3]; we identified different strategies, and preliminary performance results are quite interesting.

*bella@uniroma2.it

†sfilippone@uniroma2.it

2 The physical model

The lattice Boltzmann method takes its origin from the idea of solving the macroscopic fluid dynamics through a microscopic kinetic approach, trying to mathematically describe movements and interactions of the small particles that constitute the flow. The base equation is the Lattice Boltzmann Equation (LBE), that, established already in 1854 for the kinetic theory of ideal gases by Maxwell and Boltzmann, tells us how the particle distribution of a diluted fluid changes with time [12]:

$$\partial_t f + \xi \cdot \nabla f = \Omega \quad (1)$$

$f(\vec{x}, \vec{c}, t)$ is the particle distribution function or population that describes the probability of finding a number of particles at time t in a certain position \vec{x} and with a certain velocity \vec{c} . Moreover, the particles density in a certain position changes because particles collide with each other: this phenomenon is mathematically described by the collision function Ω .

To make a model which satisfies the assumptions made and is simple to simulate, a regular lattice is defined and the velocity space is drastically reduced to only a few discrete points by assuming that at each site the particles can only move along a finite number of directions [9].

A popular kinetic model is the Lattice Boltzmann Equation with the single relaxation time approximation. This means that, considering the distribution function as the sum of an equilibrium distribution f_i^{eq} and a non-equilibrium correction f_i^{neq} , where i refer to the i -th velocity direction, relaxation to equilibrium is governed by a single relaxation time τ . Therefore the collision operator Ω_i reads as follows:

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

The inverse of τ is named relaxation frequency, ω .

In the computational approach, the lattice Boltzmann discretization consists on a first order upwind discretization of the left side and on the selection of a lattice spacing and of a time step to provide an exact Lagrangian solution. When time increases of one unit, each particle moves from site \vec{x} to site $\vec{x} + \vec{c}_i \cdot \Delta t$ and the effect of collision is realized through the single relaxation time approximation, that is easy to compute. Lattice spacing Δx and time step Δt are selected to satisfy $\frac{\Delta x}{\Delta t} = c$, so that the discrete equation becomes:

$$f_i(\vec{x} + \vec{c}_i \cdot \Delta t, t + \Delta t) - f_i(\vec{x}, t) = \frac{\Delta t}{\tau} [(f_i(\vec{x}, t) - f_i^{eq})] \quad (3)$$

The above equation is the lattice Boltzmann equation with Bathnagar-Gross-Krook (BGK) approximation [2]. The left hand side is physically a streaming process and the right hand side models the relaxation to equilibrium. This means that the BGK scheme consists of two computational steps: the streaming step and the collision step.

```

do j = nyf,nyi,-1
do i = nxi,nxf
  if (ntype(i,j).eq.0) then
    f(2,i,j) = f(2,i,j-1)
    f(6,i,j) = f(6,i+1,j-1)
  endif
enddo
enddo

do j = nyf,nyi,-1
do i = nxf,nxi,-1
  if (ntype(i,j).eq.0) then
    f(1,i,j) = f(1,i-1,j)
    f(5,i,j) = f(5,i-1,j-1)
  endif
enddo
enddo

do j = nyi,nyf
do i = nxf,nxi,-1
  if (ntype(i,j).eq.0) then
    f(4,i,j) = f(4,i,j+1)
    f(8,i,j) = f(8,i-1,j+1)
  endif
enddo
enddo

do j = nyi,nyf
do i = nxi,nxf
  if (ntype(i,j).eq.0) then
    f(3,i,j) = f(3,i+1,j)
    f(7,i,j) = f(7,i+1,j+1)
  endif
enddo
enddo

```

Figure 1: Streaming phase initial implementation

It can be shown explicitly that the lattice Boltzmann model satisfies the Navier Stokes equations with a viscosity ν proportional to $\tau - 1/2$, that makes the LBM formally a second order scheme for solving incompressible flows [12]. This approach is compatible with the physics stating mass and momentum conservation and the macroscopic variables of the flow can be actually calculated integrating the distribution functions over the particle speed space C .

The computational advantage of the lattice Boltzmann method is the simple form of the governing equations and the easiness of programming: the streaming step takes very little computational effort and the collision step is completely local.

While in the traditional approach the convection terms of the solving equations are non-linear and Laplacian operators are used for viscosity, in the LBM convection is linear and the fluid viscosity effects are modeled through the linearized collision operator. Moreover the velocity space discretization makes the model natural to parallelize

The fact that LBE is constructed on uniform cartesian grids is a particularly severe limitation where the complex geometry of internal and external boundaries cannot be fitted by regular lattices.

3 Code optimization and parallelization

As mentioned above, the LBM implementation comprises two phases: streaming and collision.

The Collision phase lends itself very favorably to parallelization, as there are essentially no spatial dependencies among variables.

The streaming phase instead is the most critical part of the application as far as the parallelization is concerned; moreover it takes up to 25% of the total running time. A straightforward implementation of the above formula is shown in Figure 1

The streaming phase may be attacked by noting that it contains a number of dependencies that prevent an efficient parallelization in its form; however most of these difficulties may be overcome by alternating among two matrices, one for the odd steps and one for the even steps; the resulting code is:

```

do j=nyi,nyf
  do i=nxi,nxf
    if (ntype(i,j) .eq. 0) then
      f_odd(1,i,j) = f_even(1,i-1,j)
      f_odd(2,i,j) = f_even(2,i,j-1)
      f_odd(3,i,j) = f_even(3,i+1,j)
      f_odd(4,i,j) = f_even(4,i,j+1)
      f_odd(5,i,j) = f_even(5,i-1,j-1)
      f_odd(6,i,j) = f_even(6,i+1,j-1)
      f_odd(7,i,j) = f_even(7,i+1,j+1)
      f_odd(8,i,j) = f_even(8,i-1,j+1)
    end if
  enddo
enddo

```

and a simple schedule achieves a good parallelization. Despite the increase in memory occupancy, there is a substantial benefit in the cache usage pattern providing an overall 15% serial speedup.

The other technique used in this application has been to combine all phases, i.e. streaming, density and collision operators, into a single computational loop, so as to achieve maximal data reuse. With this modification the code structure becomes:

```

do istep = 1,nsteps

  if (mod(istep,2).eq.1) then
!$OMP PARALLEL DEFAULT(SHARED)

c ----- static & collide -----
  call step_complete (ff0_A,ff0_B,feqf0,rhof0,uf0,vf0,1,nx,1,ny,
.      nx,ny,nodetype_f0,omegaf0,vett_resolv,n_resolv)
c ----- Boundary conditions -----
  call mbc(ff0_B,rhof0,uf0,vf0,1,nx,1,ny,nx,ny)
c ----- Obstacle -----

  call obst_bf (ff0_B,feqf0,omegaf0,rhof0,uf0,vf0,nodetype_f0,
.      delta_f0,0,nx+1,0,ny+1,nx,ny,vett_obst,n_obst)

!$OMP END PARALLEL

```

```

        else
!$OMP PARALLEL DEFAULT(SHARED)
c ----- static & collide -----
        call step_complete(ff0_B,ff0_A,feqf0,rhof0,uf0,vf0,1,nx,1,ny,
        .      nx,ny,nodetype_f0,omegaf0,vett_resolv,n_resolv)
c ----- Boundary conditions -----
        call mbc(ff0_A,rhof0,uf0,vf0,1,nx,1,ny,nx,ny)
c ----- Obstacle -----

        call obst_bf (ff0_A,feqf0,omegaf0,rhof0,uf0,vf0,nodetype_f0,
        .      delta_f0,0,nx+1,0,ny+1,nx,ny,vett_obst,n_obst)
!$OMP END PARALLEL
        endif

```

Note that we explicitly include an obstacle for the fluid in the computational domain, thus providing for some computational imbalance.

To overcome the computational imbalance we create a specific data structure to control the work done on a specific piece of the computational domain; walking through this data structure gives a satisfactory balance, even in presence of an obstacle in the physical model. In our case we create a matrix *node_type(i,j)* specifying the type of mesh node (*i,j*); sub-routine *step_complete*, which coalesces the *move* and *streaming* phases, only needs to work on elements of type 0 (fluid).

The original code:

```

do i=1,nx
  do j=1,ny
    if (nodetype(i,j) .eq. 0 ) then
      c-----
      c---- Work with node (i,j) c-
    endif
  enddo
enddo

```

becomes:

```

do i=1,nx
  do j=1,ny
    if (nodetype(i,j) .eq. 0 ) then
      c-----
      c---- Add node (i,j) in structure of
      c---- sub-routine(s) that must work
      c---- with node (i,j) if it is of type "0".
      c-
      else if (nodetype(i,j) .eq. 1 ) then

```

```

c-----
c---- Add node (i,j) in structure of
c---- sub-routine(s) that must work
c---- with node (i,j)if it is of type "1"..
c-
endif
enddo
enddo

```

with the walk-through order determined by the memory storage layout to maximize cache hit ratio.

Each step then becomes:

```

do k=1,num_node_b2_process
c-----
c---- Take from the structure associate with this sub-routine
c---- the indexes of next node by to process
c-
c----- c---- Work with node (i,j) c-
enddo

```

where *num_node_b2_process* is the number of nodes contained in the sub-domain under consideration.

In this structure the iterations are independent of each other, and are balanced. The strategy is implemented with a simple integer array, storing the coordinates of the active mesh nodes:

```

do ij=1,n_resolv
i=v_resolv(0,ij) j=v_resolv(1,ij)
... c----- c---- Work with node (i,j) c-
enddo

```

and a similar strategy is applied to all nodes in the obstacle:

```

do ij=1,n_obst
i=v_obst(0,ij) j=v_obst(1,ij)
...
c----- c---- Work with node (i,j)
c-
enddo

```

A side effect of this approach is that there is no need to test for the mesh node type while executing the computational loop. A simple STATIC schedule is sufficient to get interesting results.

CPU's	Time	Speedup	Efficiency
1	268.0		
2	150.3	1.78	0.89
3	107.9	2.48	0.83
4	112.3	2.39	0.60
5	86.2	3.11	0.62
6	69.6	3.85	0.64

Table 1: Run with GUIDED schedule

CPU's	Time	Speedup	Efficiency
1	23.02		
2	15.82	1.455	0.7
3	11.8	1.951	0.65
4	9.93	2.318	0.58
5	8.37	2.75	0.55
6	9.09	2.532	0.42

Table 2: Run with domain 50×40 , no obstacle

4 Experimental results

Our numerical experiments have been performed on a Silicon Graphics Origin3200 with 6 CPUs MIPS R12000 running at 400 MHz, equipped with 8 GB of main memory and 8MB of L2 cache, with operating system IRIX 64 Version 6.5

The runs with the original code structure with DYNAMIC and GUIDED schedules were not very satisfactory; an example can be seen in Table 1.

The strategy outlined in Sec 3 gave much better results on a wide range of domain sizes; we show these results are shown in Tables 2, 3, 4 and 5.

As can be seen, the efficiencies are quite good; a rather counterintuitive

CPU's	Time	Speedup	Efficiency
1	188.89		
2	103.67	1.822	0.9
3	73.15	2.582	0.86
4	52.56	3.594	0.90
5	40.29	4.688	0.94
6	31.94	5.914	0.99

Table 3: Run with domain 500×400 , no obstacle

CPU's	Time	Speedup	Efficiency
1	78.7		
2	40	1.9675	0.98
3	27	2.9148	0.97
4	20.05	3.9252	0.98
5	16.74	4.7013	0.94
6	16.42	4.7929	0.80

Table 4: Run with domain 200×200 , obstacle $R = 24.9$

CPU's	Time	Speedup	Efficiency
1	177		
2	103.3	1.7135	0.86
3	67.8	2.6106	0.87
4	50.3	3.5189	0.88
5	36.26	4.8814	0.98
6	29.56	5.9878	0.998

Table 5: Run with domain 500×400 , obstacle $R = 24.9$

result, currently under investigation, is the increase in efficiency with the number of processors shown in Table 5.

5 Conclusion

We have taken a serial code implementing a Lattice-Boltzmann fluid dynamics model, and have applied to it a number of optimizations; then, we have experimented with different strategies, obtaining some very interesting results on a real application.

Future work will be devoted to refining the new dynamic pool-of-work approach; we also plan to test the effectiveness of the same algorithmic approach on shared memory PCs running with Linux, and investigating whether a hybrid MPI/OpenMP approach provides further benefits. On the application side we also plan to move to a 3D physical model.

References

- [1] R. Benzi, S. Succi, M. Vergassola, Theory and application of the lattice Boltzmann equation, Phys. Rep. 222 (3) (1992) 147.

- [2] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems, *Phys. Rev.* 94 (1954) 511.
- [3] Rohit Chandra, Dave Kohr, Ramesh Menon, Leo Dagum, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2000.
- [4] R.Cornubert, D.d’Humières, and D.Levermore. A Knudsen layer theory for lattice gases. *Physica D* 47 241.
- [5] U. Frish, B.Hasslacher, Y. Pomeau, Lattice gas cellular automata for the Navier-Stokes equations, *Phys Rev. Lett.* 56 (1986) 1505
- [6] L.-S. Luo, Theory of the Lattice Boltzmann Method: Lattice Boltzmann Models for Non-ideal Gases, ICASE Report 2001-8, NASA/CR-2001-210858 (2001).
- [7] G. Mc Namara, G. Zanetti, Use of the Boltzmann equation to simulate lattice-gas automata, *Phys. Rev. Lett.* 61 (1988), 2332.
- [8] R. Mei, L.-S. Luo, W. Shyy, An accurate curved boundary treatment in the lattice Boltzmann method, *J. Comp. Phys.* 155 (1999) 307.
- [9] Y.H. Qian, S. Succi, S. Orzag, Recent advances in lattice Boltzmann computing, *Ann. Rev. Comp. Phys.* 3 (1995) 195.
- [10] P.A. Skordos. Initial and boundary conditions for the lattice Boltzmann method. *Phys. Rev. E*, 48 (6) (1993) 4823.
- [11] S. Succi, Lattice Boltzmann equation: failure or success?, *Physica A* 240 (1997) 221.
- [12] S. Succi, The lattice Boltzmann equation for fluid dynamics and beyond, Clarendon, Oxford, 2001.
- [13] N. Takada, M. Misawa, A. Tomiyama, S. Hosokawa, Simulation of Bubble motion under Gravity by Lattice Boltzmann Method, *J. of Nuc. Sci. and Tech.* 38 (5) (2001) 330.
- [14] D.P. Ziegler, Boundary conditions for lattice Boltzmann simulations, *J. Stat. Phys.* 71 (1993) 1171.