Deliverable D3.4

1st Version of Lattice Boltzmann CFD

2022-06-28

Work package:	WP3 Development of 1st Version of Applications	
Author(s):	Gino Perna	ES
Reviewer #1	Almut Eisenträger	FRAUN
Reviewer #2	Pavlos Malakonakis	TSI
Dissemination Level	Public	
Nature	Report	

This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 955739. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Greece, Germany, Italy, Netherlands,Spain,Switzerland.

Executive Summary

Simulation of real-life phenomena are becoming increasingly useful for prediction and design capability in the Engineering Science arena. Due to the large number of equations needed to describe the multi-physics problems, these kinds of simulations require accelerations for the simulations themselves in order to obtain results and explore different solutions in a meaningful time.
The scenario is even more demanding in terms of computational resources when talking about digital-twins and efficient prototyping of the design goals.
This document aims to describe the type of mathematical approach involved, analyze the bottlenecks and provide results of first runs in a specific Computer Fluid Dynamic method called Lattice Boltzmann Method (LBM).
In D5.4, we will further improve the algorithms in order to obtain optimized performance results and provide a final version.

LBM will be briefly described in the document with all the steps performed in order to analyze the parallelizable parts of the kernel and find a strategy for implementing a HPC hybrid FPGA based LBM solver.

Table of Contents

List of Abbreviations

HPC	High Performance Computing
FPGA	Field Programmable Gate Array
CUDA	Compute Unified Device Architecture
MPI	Message Passing Interface
OpenMP	Open Multi-Processing.
OOPS	OPTIMA OPen Source Library
LBM	Lattice Boltzmann Method
CFD	Computational Fluid Dynamics

1. Introduction

Fluid dynamics is a notoriously difficult part in simulation analysis. Only a few problems can be solved analytically. Computations often are approximated at a first degree of the turbulence models and do not catch the real physics in all space but only as a mean value. Especially when coupling more physics into the same Computational Fluid Dynamics (CFD) problem (for example sound waves, radiation etc.) the behavior can run into secondary order effects (due to oscillating waves that often form vortices). These need second degree approximations which require a lot of computational resources.
This behavior is inherently nonlinear and difficult to understand quantitatively and needs a huge quantity of equations and iterations in order to be solved.

We normally describe the macroscopic state of a flow in terms of its density, ρ, and its velocity, u, both of which are functions of position and time. These functions obey a set of coupled nonlinear partial differential equations (Navier-Stokes equations).
Navier-Stokes equations are not solvable analytically. Computational methods are often the most fruitful approach to fluid dynamics problems. The traditional approach to CFD is to discretize the Navier-Stokes equations and then solve them, numerically, for the desired boundary conditions and domain.

Boltzmann Distribution on a single discretization space (Lattice)

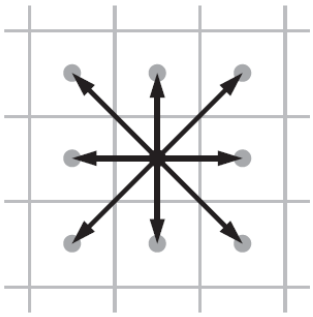
In the early 1990s, researchers developed a completely different approach to CFD, starting from physics at the molecular scale. Suppose that the flow is an ideal gas, and suppose for now that it has no macroscopic velocity ($u = 0$) and is in thermal equilibrium at temperature T. Then the molecules will have thermal velocities v that are distributed according to the Boltzmann distribution of statistical mechanics. For a two-dimensional gas, this distribution is:

$$D(\vec{v}) = \frac{m}{2\pi kT} e^{-m|\vec{v}|^2/2kT} \quad [1]$$

where m is the mass of each molecule and k is Boltzmann's constant. This is the function which, when integrated over any range of v_x and v_y , gives the probability that a particular molecule will have a velocity in that range. It is basically just a *Boltzmann factor* $e^{-E/kT}$, with E given by the ordinary kinetic energy, $0.5\,m|v|^2$. The factor in front of the exponential is needed to normalize the distribution so that its integral over *all* v_x and v_y equals 1.

In the Lattice-Boltzmann approach [1][2], we discretize both space and time so that only certain discrete velocity vectors are allowed. In this part of the project, we will use the so-called D2Q9 lattice in which there are two dimensions and just nine allowed displacement vectors, shown in the following picture. One of the allowed displacements is zero, while the

other eight take a molecule from its current site into one of the eight nearest sites in the square lattice, either horizontally, vertically, or at a 45-degree diagonal. Nine displacement vectors can be written as e_i, where i runs from 0 through 8 and each e_i has x and y components that are -1, 0, or 1, in units of the lattice spacing, Δx. If the simulation time step is Δt, then the allowed velocity vectors are given by $c \cdot e_i$, where c is an abbreviation for $\Delta x/\Delta t$.



$$\vec{e}_0 = 0$$
$$\vec{e}_1 = (1,0) \qquad \vec{e}_5 = (1,1)$$
$$\vec{e}_2 = (0,1) \qquad \vec{e}_6 = (-1,1)$$
$$\vec{e}_3 = (-1,0) \qquad \vec{e}_7 = (-1,-1)$$
$$\vec{e}_4 = (0,-1) \qquad \vec{e}_8 = (1,-1)$$

Probabilities have to be defined in these nine velocity vectors, to model the continuous Boltzmann distribution as accurately as possible. For a flow at rest (u = 0), equation [1] says that *zero* must be the most probable velocity, while the longer diagonal velocity vectors must be less probable than the shorter horizontal and vertical vectors. Velocities with the same magnitude must have the same probability, regardless of direction. The optimum probabilities turn out to be 4/9 for velocity zero, 1/9 for the four cardinal directions, and 1/36 for the diagonals. Those probabilities (or *weights*) will be marked as w_i

$$w_0 = \frac{4}{9}, \qquad w_1 = w_2 = w_3 = w_4 = \frac{1}{9}, \qquad w_5 = w_6 = w_7 = w_8 = \frac{1}{36} \qquad [2]$$

By applying the equilibrium equation and expanding equation [1] to a flow with a non-zero velocity and expanding into a Taylor series keeping up to second order degree terms, equation [1] turns into:

$$D(\vec{v}) \longrightarrow w_i \left[1 + \frac{3\,\vec{e}_i \cdot \vec{u}}{c} + \frac{9}{2}\left(\frac{\vec{e}_i \cdot \vec{u}}{c}\right)^2 - \frac{3}{2}\frac{|\vec{u}|^2}{c^2}\right]. \qquad [3]$$

where the weights ω_i are 4/9 for rest particles (i = 0), 1/9 for i = 1,2,3,4 and 1/36 for i = 5,6,7,8. Here, c is the basic speed of the lattice, u is the velocity and e_i is the direction vector for the 9 possible directions.

This expression is the key equation to be solved in the lattice defined as the collision stage (which looks for particle distributions). The LBM technique is split into two stages, collision and streaming: in the present algorithms the Bhatnagar-Gross-Krook (BGK) operator is used in order to find an equilibrium state.

Once the new distributions are calculated, they must be distributed to neighboring lattice points through the streaming step, where particle distributions are swapped between adjacent lattice points along the 8 non-zero direction vectors:

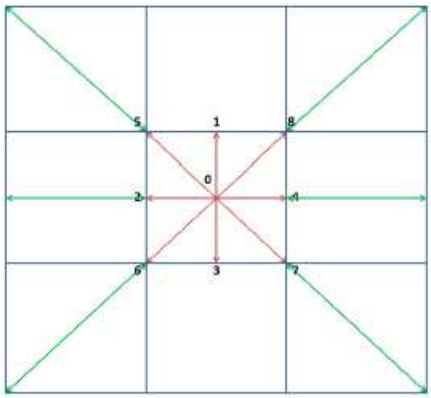


Figure 1: *Streaming Stage (in red lattice directions, in green adiacent streaming directions of particles)*

2. Definition of tasks for hardware acceleration

The initial work within the project has been to understand the timing of the mathematical kernel in order to initially focus on the most important parts.
The kernel has two important stages where calculations are involved (described here briefly in macro-areas):

- Collisions. For each lattice, do the following:
  - From the nine microscopic densities n_i, compute the macroscopic density ρ and velocity components u_x and u_y.
  - From these three macroscopic variables, use equation [3] to compute the equilibrium number densities n_i^{eq}
  - Update each of the nine number densities according to the previous step until the difference between steps becomes negligible
- Streaming. Move all the moving molecules into adjacent or diagonal lattice sites by copying the appropriate n_i values.

The problem is generally CPU bounded. Simulations are also done in the time domain, producing a time history of the flow velocities, temperatures and mass flows.

3. Implementation details

In this section we describe the main work for implementing a hybrid CPU-FPGA version of the solver mainly based on an SMP (openMP) code.

By instrumenting the kernel with run-time performance collecting libraries [3] and running a number of different simulations using D2Q9 lattice, the following timing schedule has been extracted:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.39     11.51    11.51 1168582500     0.00     0.00  computeEquilibrium
 31.68     22.43    10.92    10000     1.09     1.09  propagate
 23.09     30.39     7.96 123870000     0.00     0.00  bgk
  6.09     32.49     2.10 126067500     0.00     0.00  computeMacros
  1.96     33.16     0.68  5960000     0.00     0.00  splitEqNeq
  1.76     33.77     0.60  5960000     0.00     0.00  regularizedF
```

Figure 2: *timing framework results of a sample CFD problem*

Double and single precision versions show similar results. By analyzing the number of calls per simulation, we decided that computeEquilibrium and bgk methods should be the parts to look at.

computeEquilibrim

We initially focus on the compute Equilibrium method:

```
221
222     // compute local equilibrium from rho and u
223 double computeEquilibrium(int iPop, double rho,
224                            double ux, double uy, double uSqr)
225 {
226     double c_u = c[iPop][0]*ux + c[iPop][1]*uy;
227     return rho * t[iPop] * (
228                 1. + 3.*c_u + 4.5*c_u*c_u - 1.5*uSqr
229             );
230 }
231
```

Figure 3: *computeEquilibrium Method*

where the first part of the job has been identified to be accelerated on FPGA (lines 226 to 228).

The implementation focused on line 228 to be accelerated in hardware instead of the corresponding sequence of FP calculations.

BGK

Another method to look into was the BGK method [4]:

```
232  // bgk collision term
233  void bgk(double* fPop, void* selfData) {
234      double omega = *((double*)selfData);
235      double rho, ux, uy;
236      computeMacros(fPop, &rho, &ux, &uy);
237      double uSqr = ux*ux+uy*uy;
238      int iPop;
239      for(iPop=0; iPop<9; ++iPop) {
240          fPop[iPop] *= (1-omega);
241          fPop[iPop] += omega * computeEquilibrium (
242                                  iPop, rho, ux, uy, uSqr );
243      }
244  }
```

Figure 4: *bgk Method serial version*

The advantage of pipelining the for-loop in line 239, which in turns calls computeEquilibrum (which in turn will be fully accelerated in the second part of the project), is clear. The openMP kernel almost resembles the same pipeline as the FPGA.

4. Preliminary numerical results on JUMAX at Juelich

Preliminary tests have been performed on the JUMAX platform at Juelich by running the single thread application on the host with FPGAs comparing it with the openMP CPU version with 9 threads (nine is due to D2Q9 lattice pattern used).
The problem solved was a reference problem from our database, similar results have been achieved with other examples.

Results are summarized in the following table:

Table 1: summary of running times for 1000 time steps

Reference method	CPU [ms]	CPU+FPGA [ms]	Speedup
computeEquilibrium	11510	7947	1.44
Bgk (serial)	7960	1673	4.75
Bgk (openMP)	1245	1673	0.744

The table summarizes, in this first version, timing ratios achieved with the implementation. It should be noted that:

- The run for BGK method on FPGA was optimized for the specific lattice (it was the same for serial and openMP part on FPGA) which shows performance bottlenecks in the memory transfer part.

- BGK has been constrained to the particular 9 thread implementation (in the openMP test) in order to be comparable with the FPGA counterpart. The number nine derives from the specific pattern D2Q9 used in the tests, so the parallel tasks can be compared.

There is a small performance bottleneck in BGK mainly due to memory transfer of data buffers back and forth from/to the FPGA. All tests are performed in SP arithmetics.
The problem is time dependent and the propagate method of the code (another 30% of runtime) is relying on this.
New optimizations should be done by researching the type of buffering to optimize communications with the FPGAs and the possibility of parallelizing stripes of lattices.

5. Concluding remarks

The initial FPGA implementation of an LBM conventional method is providing promising results that encourage further optimizations. A slight performance improvement has already been experimented, if compared to the corresponding openMP part of the same code. Pipelining instructions for this type of code are not always obvious, since the flux of information is based on the type of problem to be solved.
As for all parallelization tasks, the application must be rethought (from the solver part point of view) in order to run in less time and more efficiently, with the constraint of the available resources.

6. References

[1] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Oxford University Press (2001)
[2] J.P. Rivet, J.P. Boon, *Lattice Gas Hydrodynamics* (Cambridge University Press, Cambridge, 2001)
[3] Valgrind, an instrumentation framework, last accessed 2022-06-21, https://valgrind.org/
[4] Bhatnagar–Gross–Krook operator https://en.wikipedia.org/wiki/Prabhu_Lal_Bhatnagar, last accessed 2022-06-22