CCDSC 2014

Early attempts of implementing the lattice Boltzmann method on Intel's MIC architecture

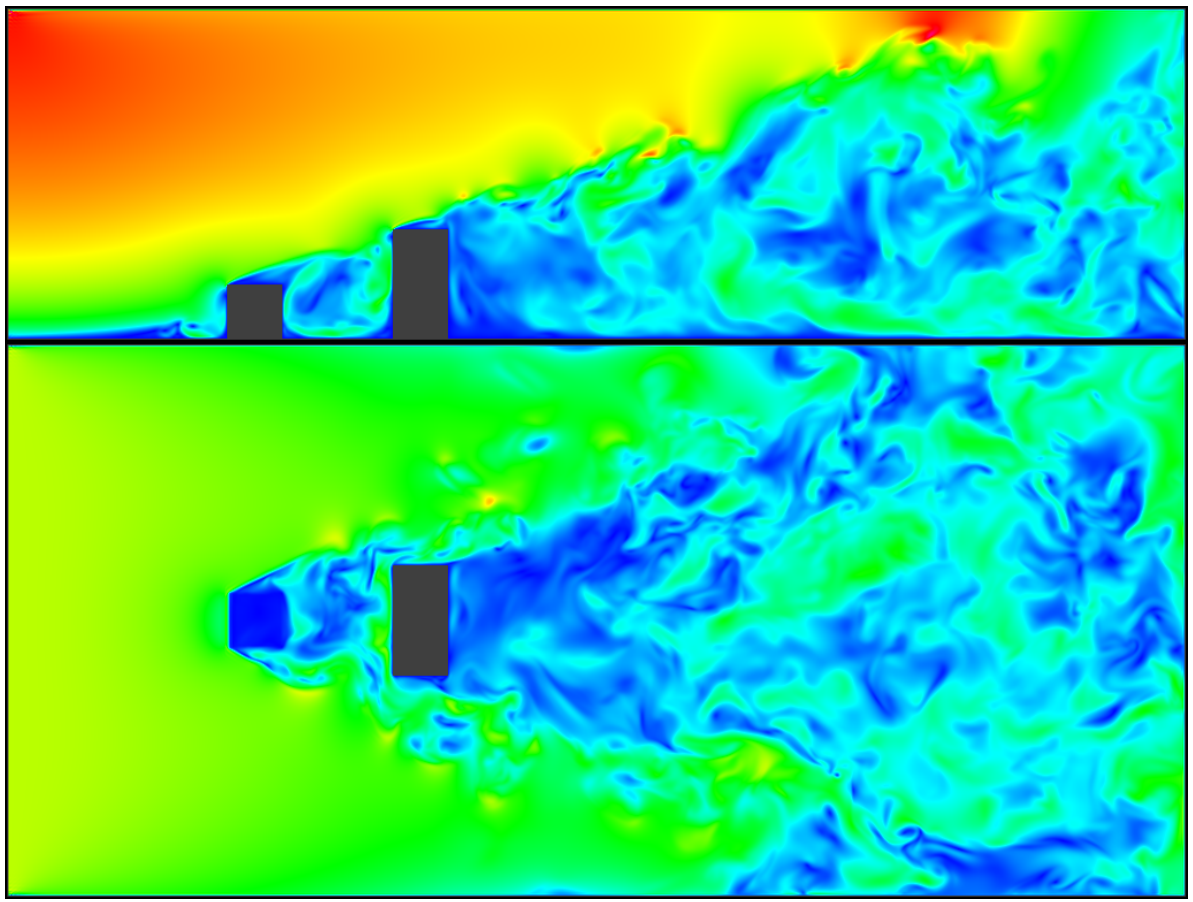
Christian Obrecht[1], Virginie Favrat[1], Frédéric Kuznik[1], Bernard Tourancheau[2]

[1] Université de Lyon, CNRS, INSA-Lyon, CETHIL UMR5008, France

[2] Université de Grenoble, UJF-Grenoble, LIG UMR5217, France

September 4, 2014

Introduction: some Coloured Fluid Dynamics

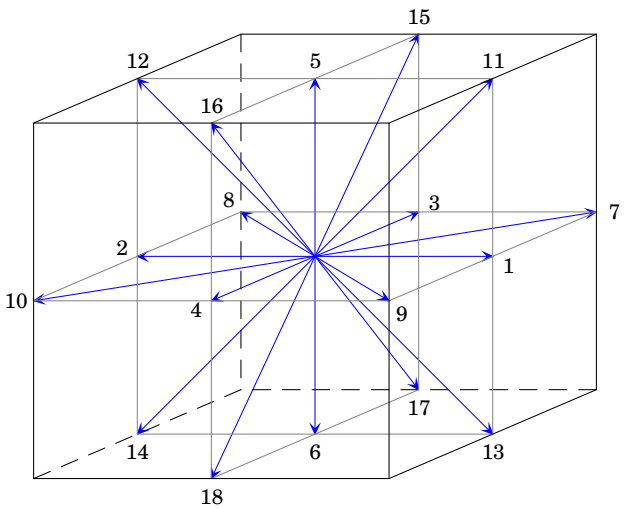


I – Lattice Boltzmann method

Lattice Boltzmann governing equation

The lattice Boltzmann method is based on a discretised version in time, space, and particle velocity space of the Boltzmann equation.

$$\left|f_\alpha(\boldsymbol{x}+\delta t\boldsymbol{\xi}_\alpha, t+\delta t)\right\rangle - \left|f_\alpha(\boldsymbol{x}, t)\right\rangle = \Omega\left(\left|f_\alpha(\boldsymbol{x}, t)\right\rangle\right)$$



$$\rho = \sum_\alpha f_\alpha$$

$$\rho\boldsymbol{u} = \sum_\alpha f_\alpha \boldsymbol{\xi}_\alpha$$

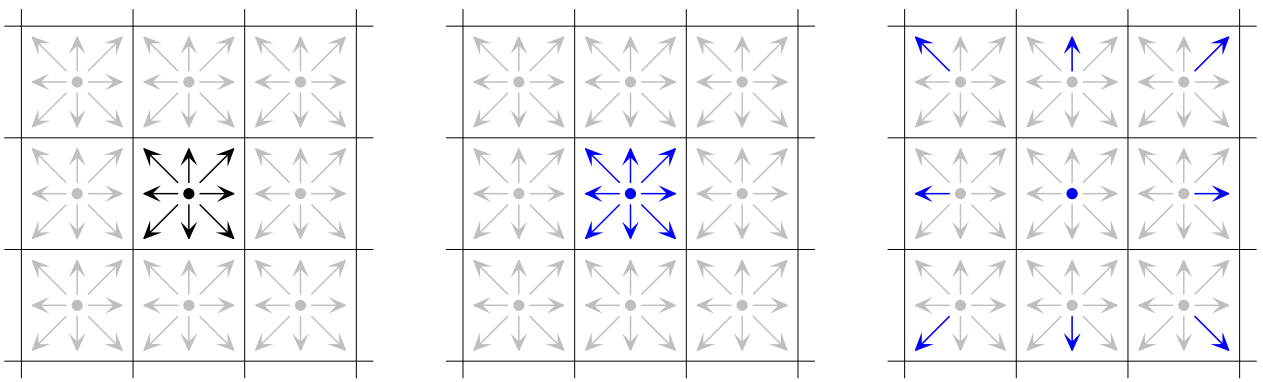
X. He and L.S. Luo.
Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation.
Physical Review E, 56(6):6811–6817, 1997.

Lattice Boltzmann algorithm

The LBM updating rule naturally splits in two steps: collision (1) and propagation (2).

$$\left|f_\alpha^*(\boldsymbol{x}, t+\delta t)\right\rangle = \left|f_\alpha(\boldsymbol{x},\ t))\right\rangle + \Omega\Big(\left|f_\alpha(\boldsymbol{x},\ t)\right\rangle\Big) \tag{1}$$

$$\left|f_\alpha(\boldsymbol{x}+\delta t\boldsymbol{\xi}_\alpha, t+\delta t)\right\rangle = \left|f_\alpha^*(\boldsymbol{x}, t+\delta t)\right\rangle \tag{2}$$



Algorithmic aspects

From an algorithmic perspective, the LBM usually:

- operates on homogeneous Cartesian meshes;
- resorts to explicit numerical schemes;
- requires only nearest neighbours synchronisation.

The LBM is therefore well-suited for parallel implementations, especially on massively parallel processors because of the regular data access pattern.

An LBM solver is a memory intensive application whose performance is generally memory-bound.

II – OpenCL implementation

Motivation

- LBM solvers based on the CUDA technology are usually able to make full profit of the computational power of Nvidia GPUs.
- CUDA is however a proprietary technology primarily designed for a specific hardware, which may become problematic over time.
- OpenCL is an interesting alternative to CUDA, since it gives access to new types of accelerators such as Intel's Xeon Phi.
- OpenCL and CUDA use similar concepts. An OpenCL version of a CUDA program may therefore follow the same design.
- When targeting GPU based accelerators, the resulting program may benefit from previously devised optimisations.

J. Tölke and M. Krafczyk.
TeraFLOP computing on a desktop PC with GPUs for 3D CFD.
International Journal of Computational Fluid Dynamics, 22(7):443–456, 2008.

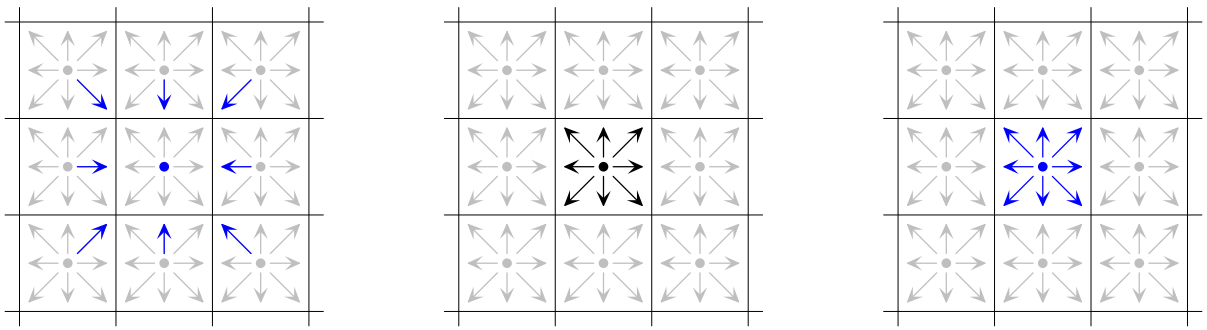
OpenCL implementation principles

1. Fuse collision and propagation into a single kernel to avoid unnecessary data transfer.
2. Map the OpenCL global domain to the lattice to take advantage of massively parallel architectures.
3. Use an appropriate data layout in order to maximise coalescent concurrent global memory accesses.
4. Launch the collision-and-propagation kernel at each time step and use two instances of the lattice to avoid synchronisation issues.

C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux.
A new approach to the lattice Boltzmann method for graphics processing units.
Computers and Mathematics with Applications, 61(12):3628–3638, 2011.

OpenCL implementation principles (continued)

5. Use in-place propagation instead of out-of-place propagation to optimise cache usage.

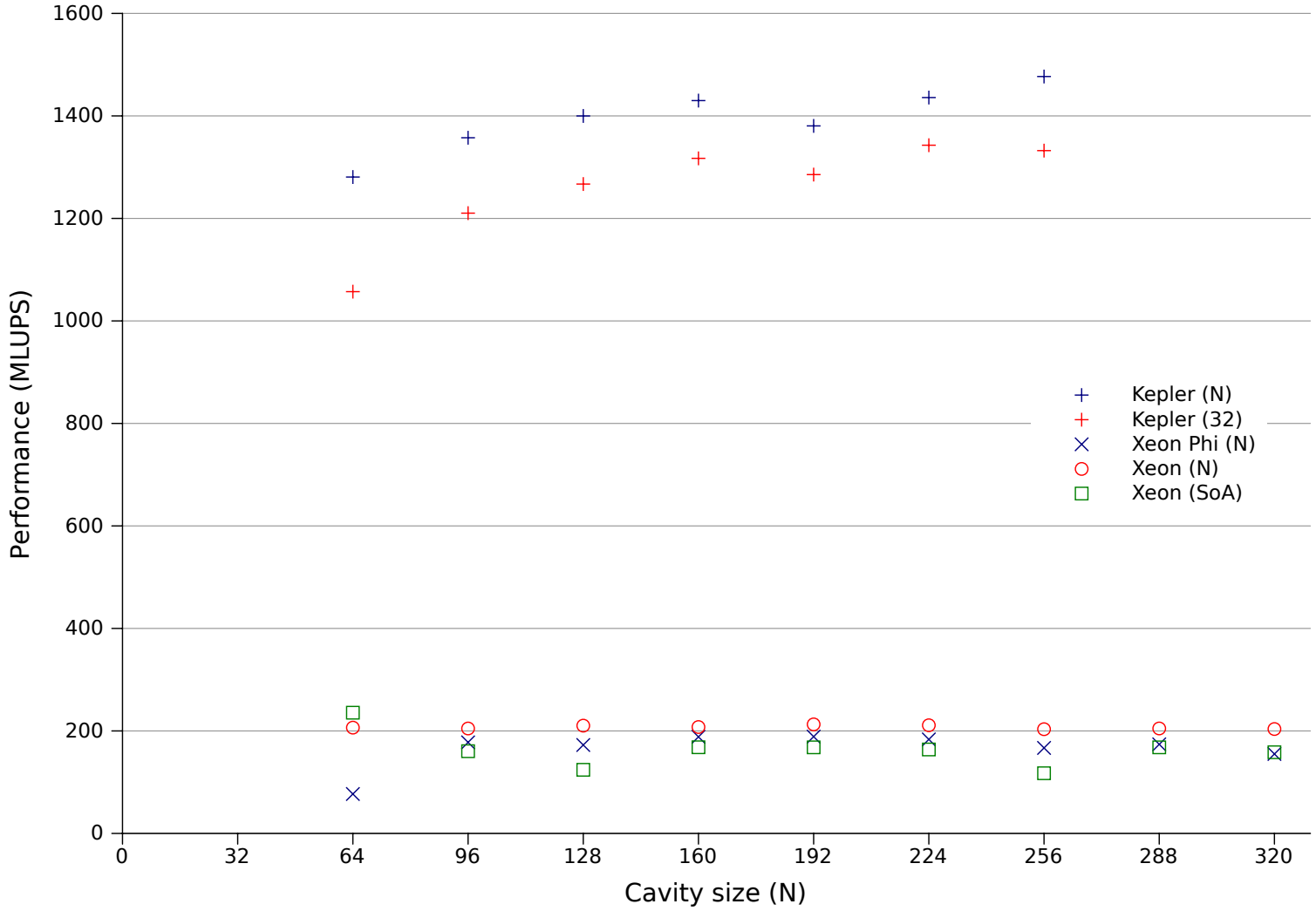


C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux.
A new approach to the lattice Boltzmann method for graphics processing units.
Computers and Mathematics with Applications, 61(12):3628–3638, 2011.

Methodology

- To evaluate our OpenCL LBM code, we performed simulations of the lid-driven cubic cavity test case.
- The program was tested on three platforms: Nvidia GeForce GTX Titan, Intel Xeon Phi 7120P, Intel Xeon E5-2670 (deca-core, double socket).
- Performance is reported in MLUPS (million lattice-node updates per second).
- The simulations were performed for increasing cavity size with various runtime configurations.

Results


Performance (MLUPS)
1600
1400
1200
1000
800
600
400
200
0
0
32
64
96
128
160
192
224
256
288
320
Cavity size (N)
Kepler (N)
Kepler (32)
Xeon Phi (N)
Xeon (N)
Xeon (SoA)

Discussion

- For all three platforms, maximal performance is achieved with one-dimensional work-groups.
- For Xeon and Xeon Phi, the size of the work-group has little impact whereas for the Kepler, best performance is reached for the largest possible size.
- Structures of arrays (SoA) are considerably more efficient than arrays of structures (AoS) for the Kepler and the Xeon Phi. The opposite conclusion holds for the Xeon.
- For the Kepler, the data throughput is on average around 212 GB/s, i.e. 89 % of the maximum, whereas the Xeon Phi reaches 25 GB/s, i.e. 10 % of the maximum.

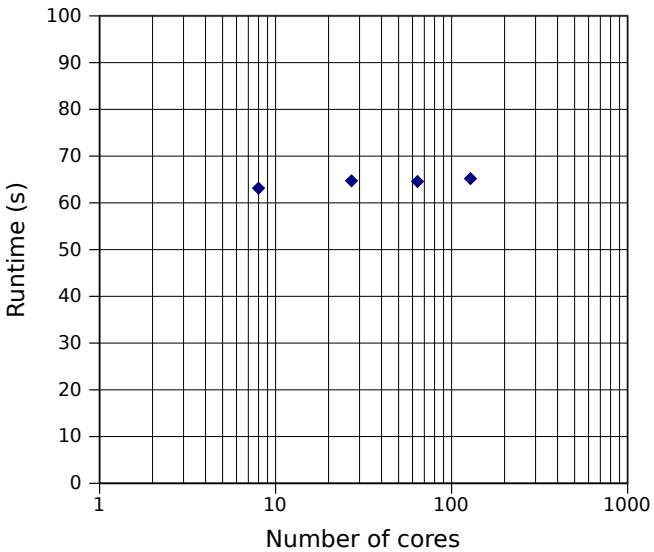
III – MPI implementation

Motivation and design

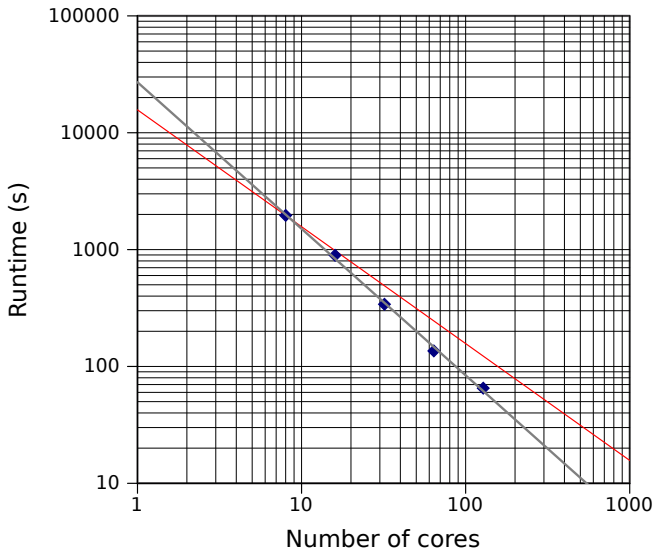
- The aim of the MPI implementation is to test a more flexible framework than OpenCL, making possible more complex data transfer between sub-domains which occur for instance in case of grid refinement.
- Each process is associated to a cuboid sub-domain. The connectivity between sub-domains is described by a JSON configuration file.
- At each time step, the out-going particular densities are copied to buffers for faces and edges then sent to the neighbouring sub-domains (non-blocking).
- Upon reception, the in-coming particular densities for faces and edges are merged into auxiliary arrays, performing partial propagation.

C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux.
Scalable Lattice Boltzmann Solvers for CUDA GPU Clusters. 39 (6-7), 259-270
Parallel Computing, 39(6-7):259–270, 2013.

Scalability on Plafrim's Fourmi cluster



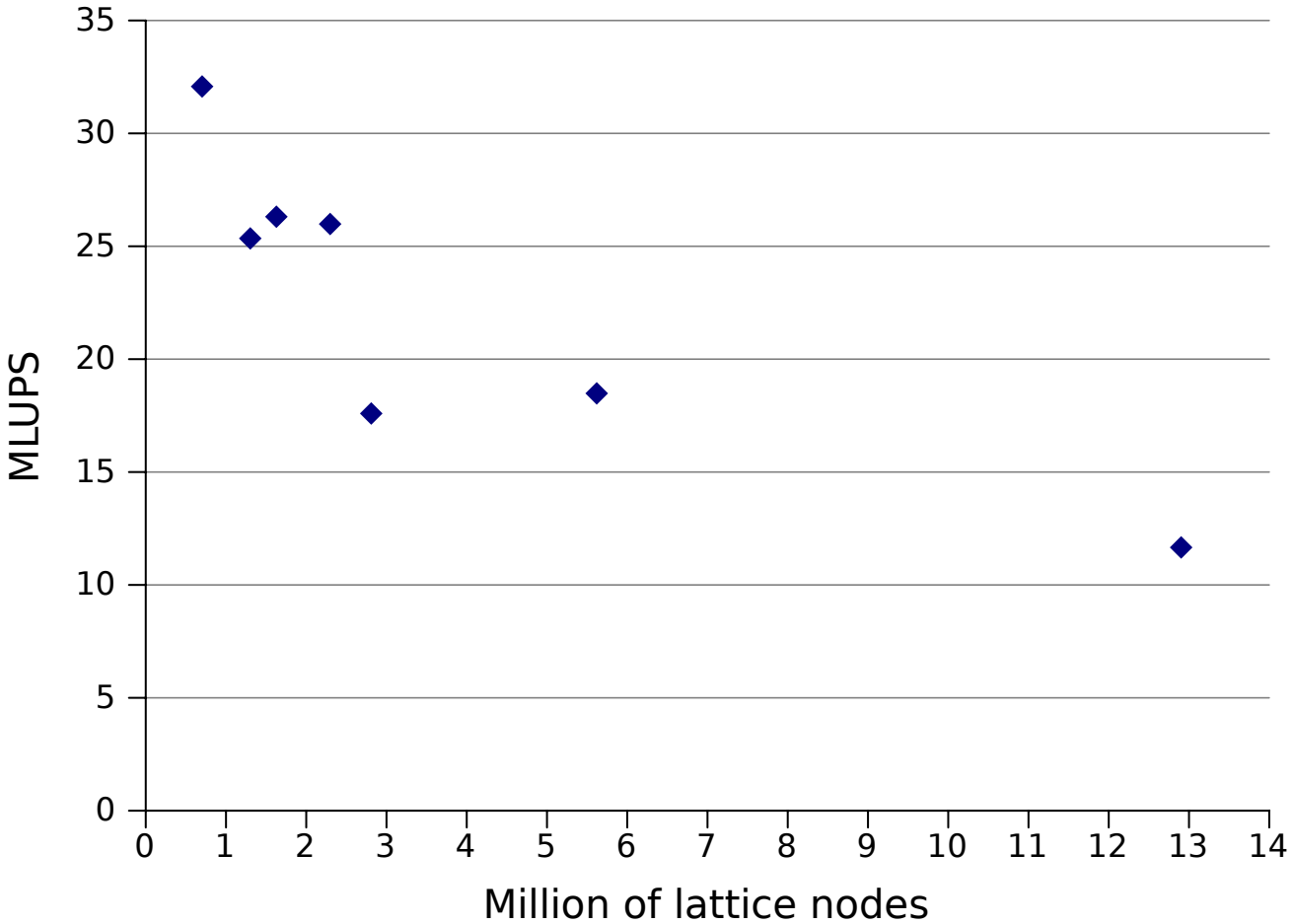
Weak scaling



Strong scaling

With 128 CPU cores, performance is about 4 MLUPS per core.

Performance on Xeon Phi



Tested on a Xeon Phi 3120A in native mode. Performance is at most 0.56 MLUPS per core.

Conclusions

- In this contribution, we present two portable implementations of the lattice Boltzmann method, based on OpenCL and MPI.
- The OpenCL version performs well on Nvidia GPU and Intel CPU.
- On the Xeon Phi, performance of the OpenCL version is below expectation, which requires further investigations.
- Performance of the MPI version of the MPI version is disappointing. Improvements will probably require the use of vectorisation intrinsics at the cost of portability.

Thank you for listening!

Acknowledgements: Intel, PlaFrim.