HPC-LEAP
EUROPEAN JOINT DOCTORATES

Heterogeneous implementation of the D2Q37 Lattice Boltzmann Method

Alessandro Gabbana

Università degli studi di Ferrara
Bergische Universität Wuppertal

September 27, 2016

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No' 642069

Outline

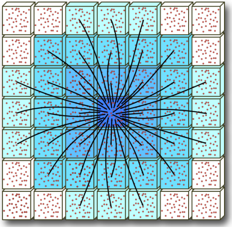
Outline

1. Lattice Boltzmann Method
    - Lattice Boltzmann Equation
    - Computational Scheme
2. Programming heterogeneous architectures
3. Data Layout Optimization
4. Load Balancing
5. Performances & Results

Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

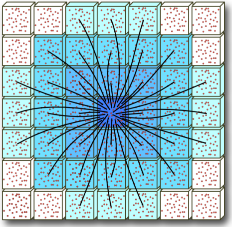
$$\vec{v} \rightarrow \{\vec{e_i}, i = 1, \ldots, m\}$$

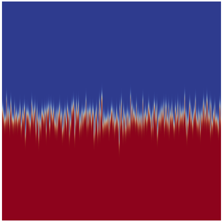


Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

$$\vec{v} \rightarrow \{\vec{e_i}, i = 1, \ldots, m\}$$

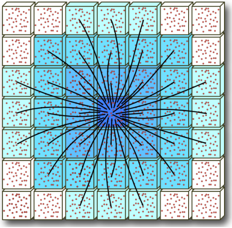


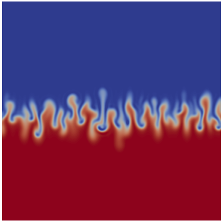


Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

$$\vec{v} \rightarrow \{\vec{e_i}, i = 1, \ldots, m\}$$

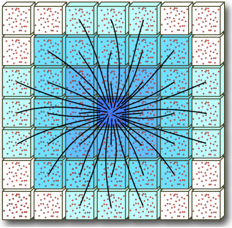


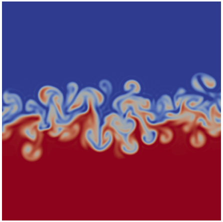


Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

$$\vec{v} \rightarrow \{\vec{e_i}, i = 1, \ldots, m\}$$

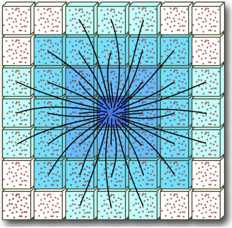


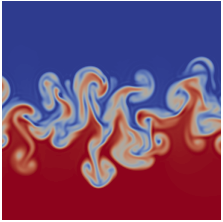


Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

$$\vec{v} \to \{\vec{e}_i, i = 1, \dots, m\}$$

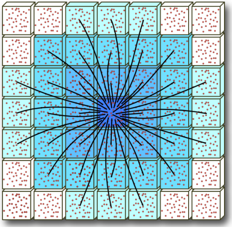


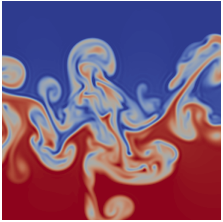


Lattice Boltzmann Method

- Lattice Boltzmann Method: Computational fluid dynamics method for solving complex fluid flows.
- Second order approximation of the **Navier-Stokes** equations.
- A set of virtual particles called **populations** arranged at the edges of a discrete regular mesh.
- Particles only have a finite number of velocity directions:

$$\vec{v} \rightarrow \{\vec{e_i}, i = 1, \dots, m\}$$





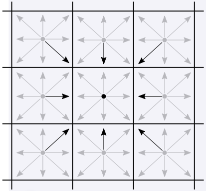
Lattice Boltzmann Equation

$$f_i(x + e_i \Delta t, t + \Delta t) = f_i(x,t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(x,t) - f_i(x,t)\right), \ i = 1 \dots m$$

Lattice Boltzmann Equation

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x, t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(x, t) - f_i(x, t)\right), \; i = 1 \ldots m$$

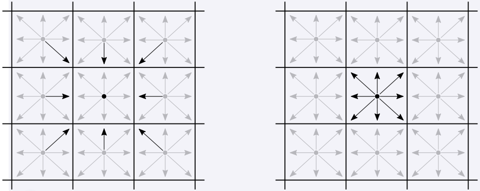
$$\tilde{f}_i(x, t) = f_i(x - e_i\Delta t, t), \quad i = 1 \ldots m$$



Lattice Boltzmann Equation

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x, t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(x, t) - f_i(x, t)\right), \; i = 1 \dots m$$

$$\tilde{f}_i(x, t) = f_i(x - e_i\Delta t, t), \quad i = 1 \dots m$$

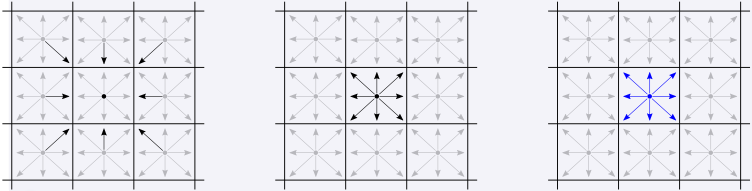


Lattice Boltzmann Equation

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x, t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(x, t) - f_i(x, t)\right), \; i = 1 \ldots m$$

$$\tilde{f}_i(x, t) = f_i(x - e_i\Delta t, t), \quad i = 1 \ldots m$$

$$f_i(x, t + \Delta t) = \tilde{f}_i(x, t) + \frac{\Delta t}{\tau}\left(f_i^{eq}(x, t) - \tilde{f}_i(x, t)\right), \; i = 1 \ldots m$$



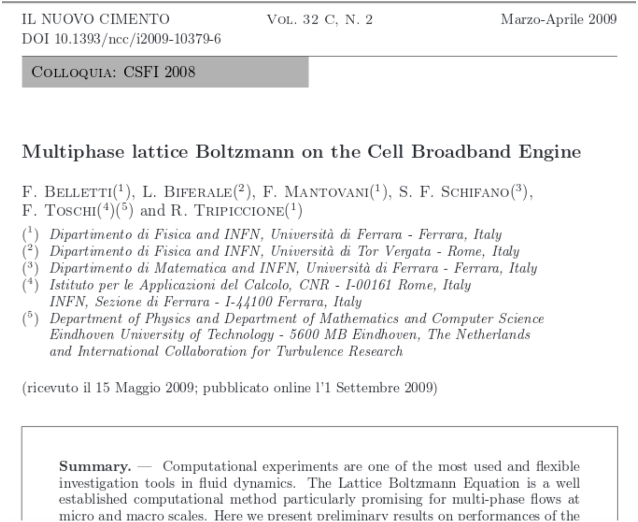
Computational Scheme

```
for all time step do
    $<$ Set boundary conditions $>$
    for all lattice site do {in parallel}
        $<$ Propagate $>$
        $<$ Collide $>$
    end for
end for
```

- In principle *propagate* and *collide* could be fused.
- Convenient for benchmarking to keep the memory bound section separated from the compute intensive one.

Outline

Many architectures... many implementations!



IL NUOVO CIMENTO   VOL. 32 C, N. 2   Marzo-Aprile 2009
DOI 10.1393/ncc/i2009-10379-6

COLLOQUIA: CSFI 2008

Multiphase lattice Boltzmann on the Cell Broadband Engine

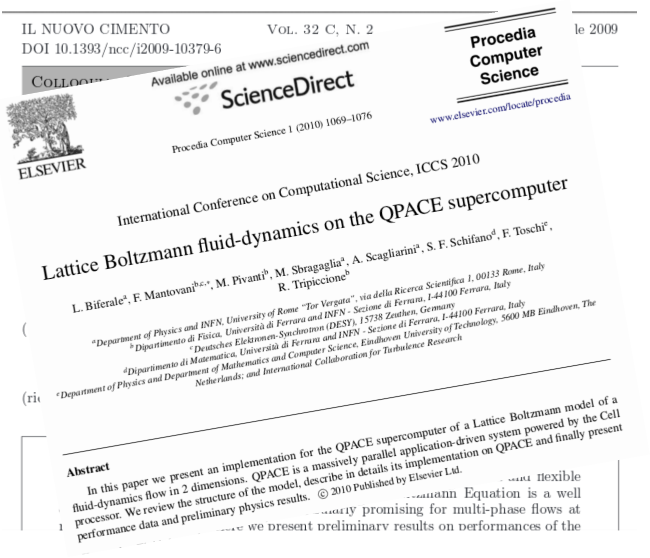
F. BELLETTI(1), L. BIFERALE(2), F. MANTOVANI(1), S. F. SCHIFANO(3),
F. TOSCHI(4)(5) and R. TRIPICCIONE(1)

(1) *Dipartimento di Fisica and INFN, Università di Ferrara - Ferrara, Italy*
(2) *Dipartimento di Fisica and INFN, Università di Tor Vergata - Rome, Italy*
(3) *Dipartimento di Matematica and INFN, Università di Ferrara - Ferrara, Italy*
(4) *Istituto per le Applicazioni del Calcolo, CNR - I-00161 Rome, Italy*
INFN, Sezione di Ferrara - I-44100 Ferrara, Italy
(5) *Department of Physics and Department of Mathematics and Computer Science*
Eindhoven University of Technology - 5600 MB Eindhoven, The Netherlands
and International Collaboration for Turbulence Research

(ricevuto il 15 Maggio 2009; pubblicato online l'1 Settembre 2009)

Summary. — Computational experiments are one of the most used and flexible investigation tools in fluid dynamics. The Lattice Boltzmann Equation is a well established computational method particularly promising for multi-phase flows at micro and macro scales. Here we present preliminary results on performances of the

Many architectures... many implementations!



IL NUOVO CIMENTO Vol. 32 C, N. 2 le 2009
DOI 10.1393/ncc/i2009-10379-6

Available online at www.sciencedirect.com
ScienceDirect

Procedia Computer Science 1 (2010) 1069–1076

Procedia Computer Science

www.elsevier.com/locate/procedia

ELSEVIER

International Conference on Computational Science, ICCS 2010

Lattice Boltzmann fluid-dynamics on the QPACE supercomputer

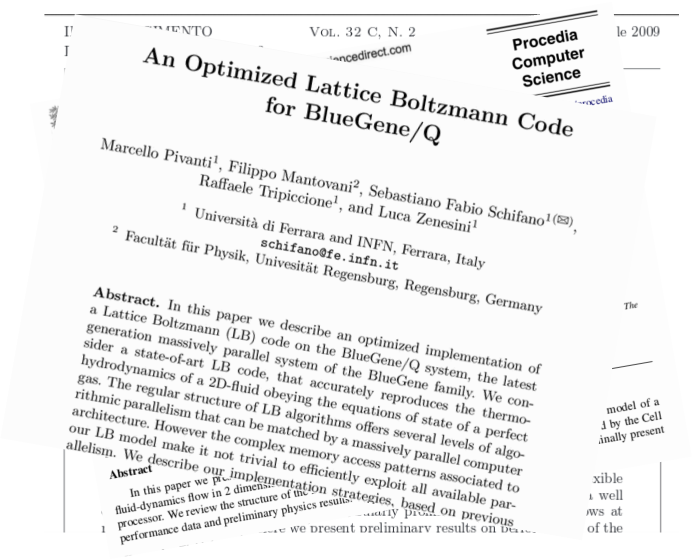
L. Biferale[a], F. Mantovani[b,c,*], M. Pivanti[b], M. Sbragaglia[a], A. Scagliarini[a], S. F. Schifano[d], F. Toschi[e],
R. Tripiccione[b]

[a] Department of Physics and INFN, University of Rome "Tor Vergata", via della Ricerca Scientifica 1, 00133 Rome, Italy
[b] Dipartimento di Fisica, Università di Ferrara and INFN - Sezione di Ferrara, I-44100 Ferrara, Italy
[c] Deutsches Elektronen-Synchrotron (DESY), 15738 Zeuthen, Germany
[d] Dipartimento di Matematica, Università di Ferrara and INFN - Sezione di Ferrara, I-44100 Ferrara, Italy
[e] Department of Physics and Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands; and International Collaboration for Turbulence Research

Abstract

In this paper we present an implementation for the QPACE supercomputer of a Lattice Boltzmann model of a fluid-dynamics flow in 2 dimensions. QPACE is a massively parallel application-driven system powered by the Cell processor. We review the structure of the model, describe in details its implementation on QPACE and finally present performance data and preliminary physics results. © 2010 Published by Elsevier Ltd.

Many architectures... many implementations!



VOL. 32 C, N. 2

Procedia Computer Science

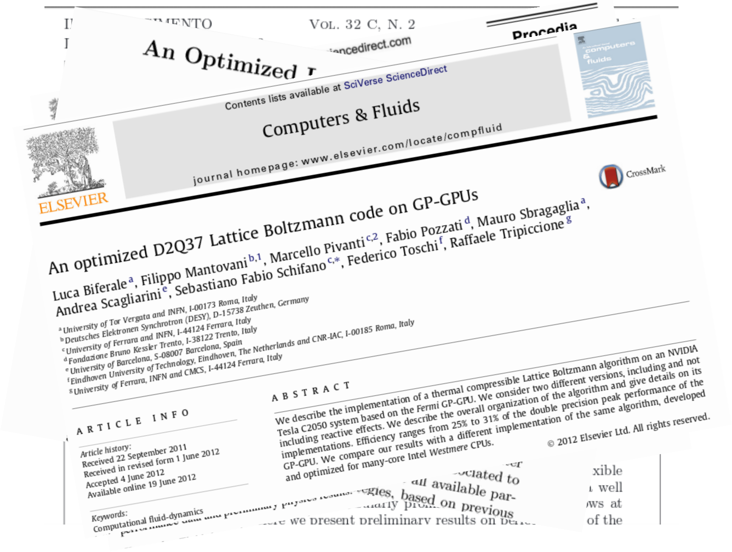
An Optimized Lattice Boltzmann Code for BlueGene/Q

Marcello Pivanti[1], Filippo Mantovani[2], Sebastiano Fabio Schifano[1](✉), Raffaele Tripiccione[1], and Luca Zenesini[1]

[1] Università di Ferrara and INFN, Ferrara, Italy
`schifano@fe.infn.it`
[2] Facultät für Physik, Univesität Regensburg, Regensburg, Germany

Abstract. In this paper we describe an optimized implementation of a Lattice Boltzmann (LB) code on the BlueGene/Q system, the latest generation massively parallel system of the BlueGene family. We consider a state-of-art LB code, that accurately reproduces the thermo-hydrodynamics of a 2D-fluid obeying the equations of state of a perfect gas. The regular structure of LB algorithms offers several levels of algorithmic parallelism that can be matched by a massively parallel computer architecture. However the complex memory access patterns associated to our LB model make it not trivial to efficiently exploit all available parallelism. We describe our implementation strategies, based on previous

Many architectures... many implementations!



Contents lists available at SciVerse ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

An optimized D2Q37 Lattice Boltzmann code on GP-GPUs

Luca Biferale[a], Filippo Mantovani[b,1], Marcello Pivanti[c,2], Fabio Pozzati[d], Mauro Sbragaglia[a], Andrea Scagliarini[e], Sebastiano Fabio Schifano[c,*], Federico Toschi[f], Raffaele Tripiccione[g]

[a] University of Tor Vergata and INFN, I-00173 Roma, Italy
[b] Deutsches Elektronen Synchrotron (DESY), D-15738 Zeuthen, Germany
[c] University of Ferrara and INFN, I-44124 Ferrara, Italy
[d] Fondazione Bruno Kessler Trento, I-38122 Trento, Italy
[e] University of Barcelona, S-08007 Barcelona, Spain
[f] Eindhoven University of Technology, Eindhoven, The Netherlands and CNR-IAC, I-00185 Roma, Italy
[g] University of Ferrara, INFN and CMCS, I-44124 Ferrara, Italy

ARTICLE INFO

Article history:
Received 22 September 2011
Received in revised form 1 June 2012
Accepted 4 June 2012
Available online 19 June 2012

Keywords:
Computational fluid-dynamics

ABSTRACT

We describe the implementation of a thermal compressible Lattice Boltzmann algorithm on an NVIDIA Tesla C2050 system based on the *Fermi* GP-GPU. We consider two different versions, including and not including reactive effects. We describe the overall organization of the algorithm and give details on its implementations. Efficiency ranges from 25% to 31% of the double precision peak performance of the GP-GPU. We compare our results with a different implementation of the same algorithm, developed and optimized for many-core Intel *Westmere* CPUs.

Many architectures... many implementations!

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 551 – 560

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Early experience on porting and running a Lattice Boltzmann code on the Xeon-Phi co-processor

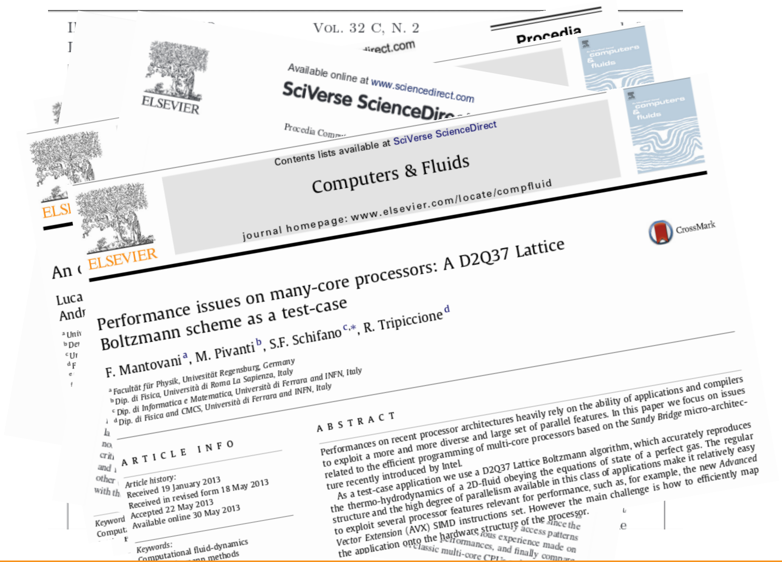
G. Crimi[a], F. Mantovani[b], M. Pivanti[d], S. F. Schifano[c,1], R. Tripiccione[c]

[a]Università di Ferrara, ITALY
[b]Facultät für Physik, Univesität Regensburg, GERMANY
[c]Dip. di Fisica e Scienze della Terra and CMCS, Università di Ferrara and INFN, ITALY
[d]Dip. di Fisica, Università di Roma La Sapienza and INFN, ITALY
[e]Dip. di Matematica e Informatica, Università di Ferrara and INFN, ITALY

Abstract

In this paper we report on our early experience on porting, optimizing and benchmarking a Lattice Boltzmann (LB) code on the Xeon-Phi co-processor, the first generally available version of the new *Many Integrated Core* (MIC) architecture, developed by Intel. We consider as a test-bed a state-of-the-art LB model, that accurately reproduces the thermo-hydrodynamics of a 2D-fluid obeying the equations of state of a perfect gas. The regular structure of LB algorithms makes it relatively easy to identify a large degree of available parallelism. However, mapping a large fraction of this parallelism onto this new class of processors is not straightforward. The D2Q37 LB algorithm considered in this paper is an appropriate test-bed for this architecture since the critical computing kernels require high performances both in terms of memory bandwidth for sparse memory access patterns and number crunching capability. We describe our implementation of the code, that builds on previous experience made on other (simpler) many-core processors and GPUs, present benchmark results and measure performances, and finally compare with the results obtained by previous implementations developed on state-of-the-art classic multi-core CPUs and GP-GPUs.

Many architectures... many implementations!



VOL. 32 C, N. 2

Available online at www.sciencedirect.com

Contents lists available at SciVerse ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

Performance issues on many-core processors: A D2Q37 Lattice Boltzmann scheme as a test-case

F. Mantovani[a], M. Pivanti[b], S.F. Schifano[c,*], R. Tripiccione[d]

[a] Facultät für Physik, Univesität Regensburg, Germany
[b] Dip. di Fisica, Università di Roma La Sapienza, Italy
[c] Dip. di Informatica e Matematica, Università di Ferrara and INFN, Italy
[d] Dip. di Fisica and CMCS, Università di Ferrara and INFN, Italy

ARTICLE INFO

Article history:
Received 19 January 2013
Received in revised form 18 May 2013
Accepted 22 May 2013
Available online 30 May 2013

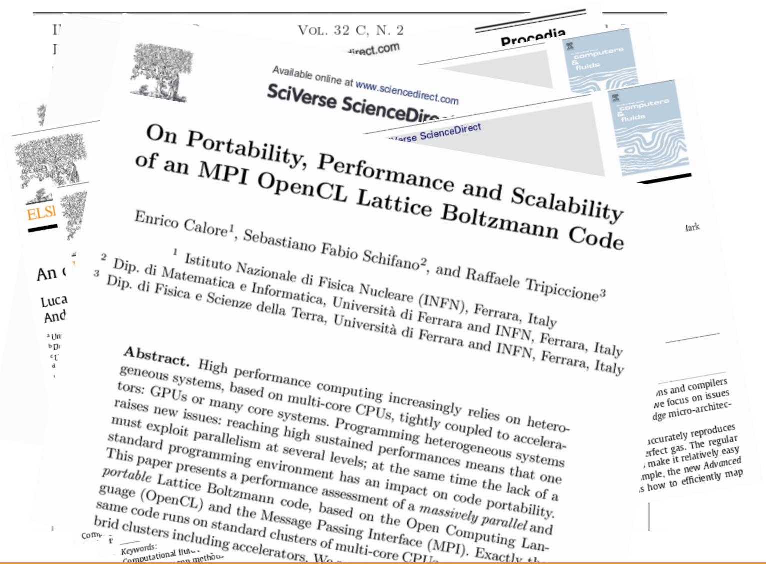
Keywords:
Computational fluid-dynamics

ABSTRACT

Performances on recent processor architectures heavily rely on the ability of applications and compilers to exploit a more and more diverse and large set of parallel features. In this paper we focus on issues related to the efficient programming of multi-core processors based on the *Sandy Bridge* micro-architecture recently introduced by Intel.

As a test-case application we use a D2Q37 Lattice Boltzmann algorithm, which accurately reproduces the thermo-hydrodynamics of a 2D-fluid obeying the equations of state of a perfect gas. The regular structure and the high degree of parallelism available in this class of applications make it relatively easy to exploit several processor features relevant for performance, such as, for example, the new *Advanced Vector Extension* (AVX) SIMD instructions set. However the main challenge is how to efficiently map the application onto the hardware structure of the processor.

Many architectures... many implementations!



VOL. 32 C, N. 2

Available online at www.sciencedirect.com

SciVerse ScienceDir

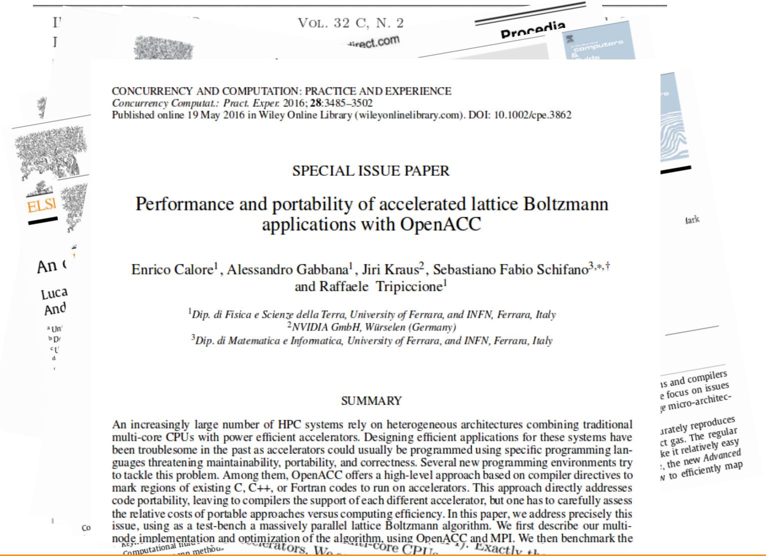
On Portability, Performance and Scalability of an MPI OpenCL Lattice Boltzmann Code

Enrico Calore[1], Sebastiano Fabio Schifano[2], and Raffaele Tripiccione[3]

[1] Istituto Nazionale di Fisica Nucleare (INFN), Ferrara, Italy
[2] Dip. di Matematica e Informatica, Università di Ferrara and INFN, Ferrara, Italy
[3] Dip. di Fisica e Scienze della Terra, Università di Ferrara and INFN, Ferrara, Italy

Abstract. High performance computing increasingly relies on heterogeneous systems, based on multi-core CPUs, tightly coupled to accelerators: GPUs or many core systems. Programming heterogeneous systems raises new issues: reaching high sustained performances means that one must exploit parallelism at several levels; at the same time the lack of a standard programming environment has an impact on code portability. This paper presents a performance assessment of a *massively parallel* and *portable* Lattice Boltzmann code, based on the Open Computing Language (OpenCL) and the Message Passing Interface (MPI). Exactly th
same code runs on standard clusters of multi-core CPU
brid clusters including accelerators. We

Many architectures... many implementations!



CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **28**:3485–3502
Published online 19 May 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3862

SPECIAL ISSUE PAPER

Performance and portability of accelerated lattice Boltzmann applications with OpenACC

Enrico Calore[1], Alessandro Gabbana[1], Jiri Kraus[2], Sebastiano Fabio Schifano[3,*,†] and Raffaele Tripiccione[1]

[1]*Dip. di Fisica e Scienze della Terra, University of Ferrara, and INFN, Ferrara, Italy*
[2]*NVIDIA GmbH, Würselen (Germany)*
[3]*Dip. di Matematica e Informatica, University of Ferrara, and INFN, Ferrara, Italy*

SUMMARY

An increasingly large number of HPC systems rely on heterogeneous architectures combining traditional multi-core CPUs with power efficient accelerators. Designing efficient applications for these systems have been troublesome in the past as accelerators could usually be programmed using specific programming languages threatening maintainability, portability, and correctness. Several new programming environments try to tackle this problem. Among them, OpenACC offers a high-level approach based on compiler directives to mark regions of existing C, C++, or Fortran codes to run on accelerators. This approach directly addresses code portability, leaving to compilers the support of each different accelerator, but one has to carefully assess the relative costs of portable approaches versus computing efficiency. In this paper, we address precisely this issue, using as a test-bench a massively parallel lattice Boltzmann algorithm. We first describe our multi-node implementation and optimization of the algorithm, using OpenACC and MPI. We then benchmark the

Solutions for performance portability in HPC

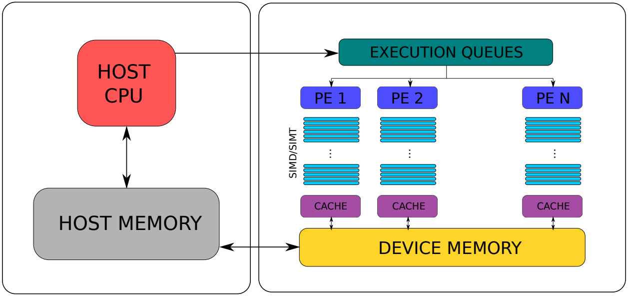
Goal

Can we have a single performance-portable code capable of running efficiently on recent heterogeneous architectures?

Tested solutions

- OpenCL
  - Low level approach
  - Future support for GPUs uncertain
- Directive based programming models
  - High level programming approach.
  - Portability becomes a duty of the compiler.
  - Several standards (OpenMP4.x, OpenACC).

Accelerator-based programming model



- Host-centric model
- Abstraction supporting both many-core (GPUs, MIC) and multi-core architectures.

Strategies for accelerator-based implementations

Two possible approaches

1. Map compute intensive sections onto the device
2. Heterogeneous implementation

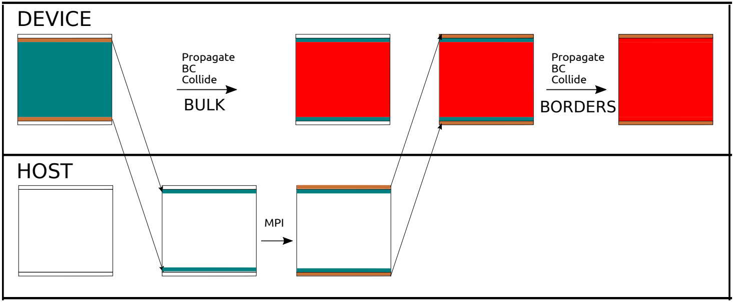
Implementation aspects common to both approaches:

- Full-matrix lattice representation with equidistant Cartesian coordinates
- Two copies of the lattice stored in memory in order to avoid data dependencies
- External halo-layers used to implement boundary conditions
- Control on data movements

Strategies for accelerator-based implementations

Offload computational intensive sections onto the device

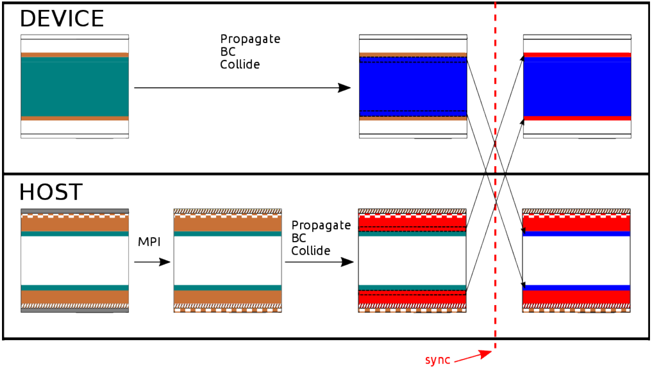
- Pros (I) : Overlap communications with computation.
- Pros (II): Needs to optimize only the code targeting the accelerator.
- Cons: The host sits idle for most of the simulation time.



Strategies for accelerator-based implementations

Heterogeneous implementation

- Pros: Fully exploits compute capabilities of the cluster.
- Cons (I) : Need for a data-layout optimizing performances on both host and accelerator.
- Cons (II): Requires load-balancing.



Outline

1 Lattice Boltzmann Method

2 Programming heterogeneous architectures

3 Data Layout Optimization
- Array of Structures (AoS)
- Structure of Arrays (SoA)
- Clusterized Structure of Arrays (CSoA)
- Clusterized Array of Structure of Arrays (CAoSoA)

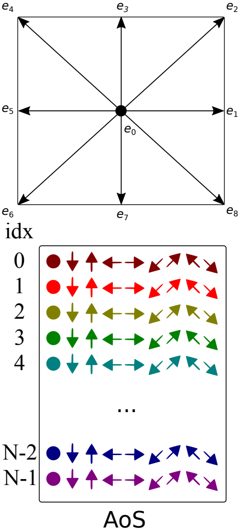
4 Load Balancing

5 Performances & Results

Array of Structures (AoS)

```
// AoS data—type definition

#define N (LX*LY)
typedef struct {
  data_t p0; // population 0
  data_t p1; // population 1
  ...
  data_t p8; // population 8
} pop_t;

aos_t lattice[N];
```

```
// snippet of collide code kernel
// computing density rho

idx = (ix*NY)+iy;

rho = 0.0;

for(p = 0; p < NPOP; p++)
  rho = rho + lattice[ NPOP*idx + p ];
```



Array of Structures (AoS): Performances

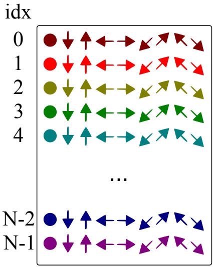
	Data Structure			
Architecture	AoS			
Haswell	10.17			
Broadwell	18.91			
Xeon Phi	21.58			
Tesla K80	16.06			
AMD Hawaii	6.07			

- Performance figures in term of MLUPS (Million Lattice Update Per Second).
- Simulations performed on a 2160x8192 lattice.

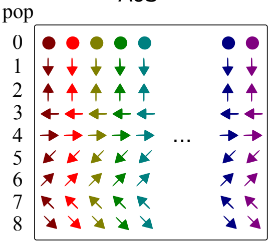
Structure of Arrays (SoA)

```
// SoA data-type definition

#define N (LX*LY)
typedef struct {
  data_t p0[N]; // population 0
  data_t p1[N]; // population 1
  ...
  data_t p8[N]; // population 8
} pop_t;

soa_t lattice;
```

```
// snippet of collide code kernel
// computing density rho

idx = (ix*NY)+iy;

rho = 0.0;

for(p = 0; p < NPOP; p++)
  rho = rho + lattice[ (p*NX*NY) + idx ];
```



AoS



SoA

Structure of Arrays (SoA): Performances

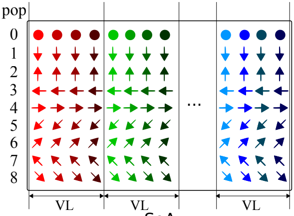
	Data Structure			
Architecture	AoS	**SoA**		
Haswell	10.17	**9.53**		
Broadwell	18.91	**16.60**		
Xeon Phi	21.58	**9.68**		
Tesla K80	16.06	**77.97**		
AMD Hawaii	6.07	**16.14**		

- Performance figures in term of MLUPS (Million Lattice Update Per Second).
- Simulations performed on a 2160x8192 lattice.

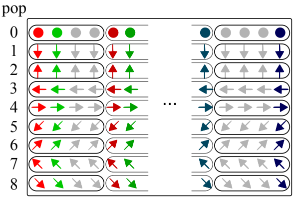
Clusterized Structure of Arrays (CSoA)

```
// cluster definition
typedef struct {
  data_t c[VL];
} vdata_t;

// CSoA data-type definition
typedef struct {
  vdata_t p[NPOP][NX*(NY / VL)];
} csoa_t;

csoa_t lattice;
```

```
// snippet of collide code kernel
// computing density rho
idx = ix*(NY / VL) + iy;

#pragma omp simd
for(k = 0; k < VL; k++){
  rho.c[k] = 0.0;
}

#pragma unroll novector
for (p = 0; p < NPOP; p++){
  #pragma omp simd
  for(k = 0; k < VL; k++)
    rho.c[k] += lattice->p[p][idx].c[k];
}
```



SoA



CSoA

Clusterized Structure of Arrays (CSoA): Performances

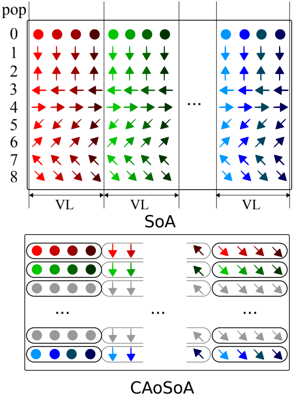
	Data Structure		
Architecture	AoS	SoA	**CSoA**
Haswell	10.17	9.53	**16.22**
Broadwell	18.91	16.60	**26.69**
Xeon Phi	21.58	9.68	**30.28**
Tesla K80	16.06	77.97	**80.34**
AMD Hawaii	6.07	16.14	**32.74**

- Performance figures in term of MLUPS (Million Lattice Update Per Second).
- Simulations performed on a 2160x8192 lattice.

Clusterized Array of Structure of Arrays (CAoSoA)

```
// CSoA data-type definition
typedef struct {
  data_t c[VL];
} vdata_t;

typedef struct {
   vdata_t p[NPOP];
} caosoa_t;

caosoa_t lattice[NX*(NY / VL)];
```

```
// snippet of collide code kernel
// computing density rho
idx = ix*(NY / VL) + iy;

#pragma omp simd
for(k = 0; k < VL; k++){
  rho.c[k] = 0.0;
}

#pragma unroll novector
for (p = 0; p < NPOP; p++){
  #pragma omp simd
  for(k = 0; k < VL; k++)
    rho.c[k] += lattice[idx].p[p].c[k];
}
```



Clusterized Array of Structure of Arrays (CAoSoA): Performances

	Data Structure			
Architecture	AoS	SoA	CSoA	**CAoSoA**
Haswell	10.17	9.53	16.22	**17.68**
Broadwell	18.91	16.60	26.69	**31.10**
Xeon Phi	21.58	9.68	30.28	**40.41**
Tesla K80	16.06	77.97	80.34	**80.19**
AMD Hawaii	6.07	16.14	32.74	**36.65**

- Performance figures in term of MLUPS (Million Lattice Update Per Second).
- Simulations performed on a 2160x8192 lattice.

Outline

1. Lattice Boltzmann Method
2. Programming heterogeneous architectures
3. Data Layout Optimization
4. **Load Balancing**
5. Performances & Results

Load Balancing

$$T_{\mathtt{exe}} = \max\{T_{\mathtt{acc}}, T_{\mathtt{host}} + T_{\mathtt{mpi}}\} + T_{\mathtt{swap}}$$

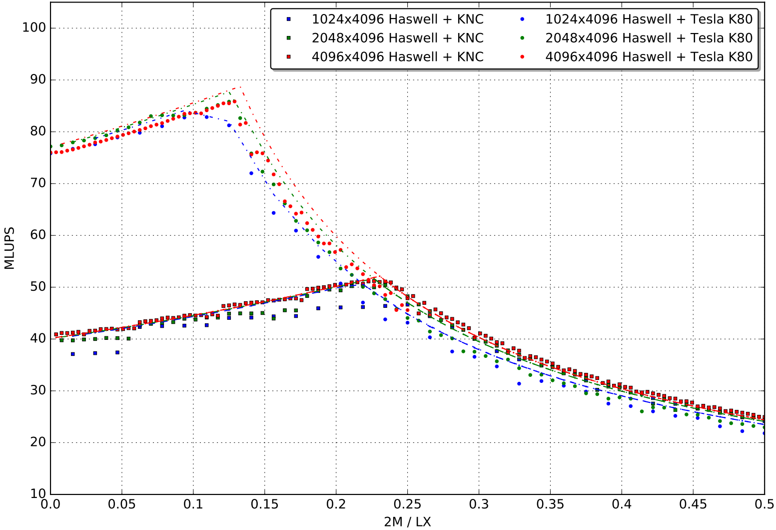
$$\begin{aligned} T_{\mathtt{acc}} &\propto (LX - 2M)LY \cdot \tau_d \\ T_{\mathtt{host}} &\propto (2M)LY \cdot \tau_h \\ T_{\mathtt{mpi}} &\propto \tau_c \text{ (constant since we are using a 1D partitioning)} \end{aligned}$$

Autotuning:

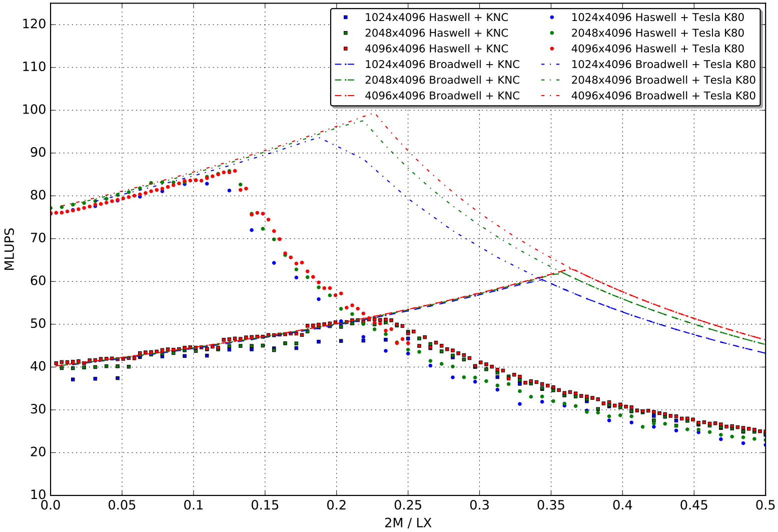
1. Get an estimate for τ_d, τ_h, τ_c by mini-benchmarks
2. Compute M such that T_{exe} by solving

$$T_{acc}(M) = T_{host}(M) + T_{mpi}(M)$$

Load Balancing: Testing the Model



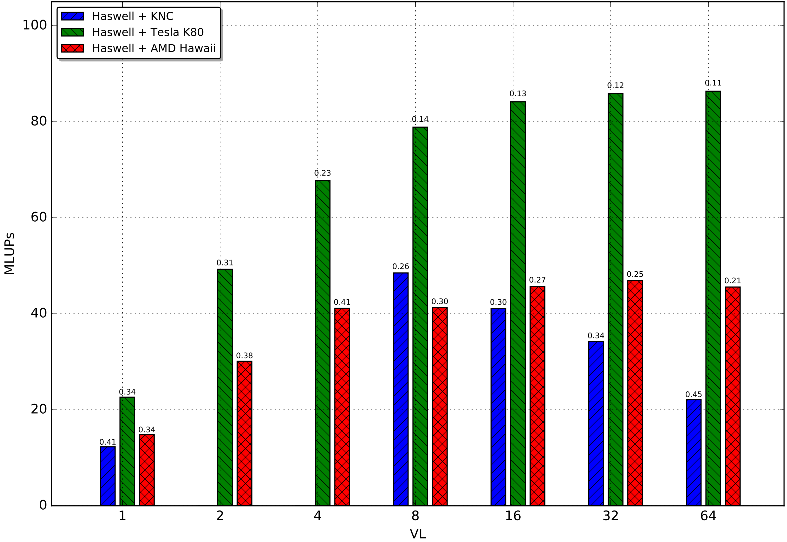
Load Balancing: Performance Forecasting



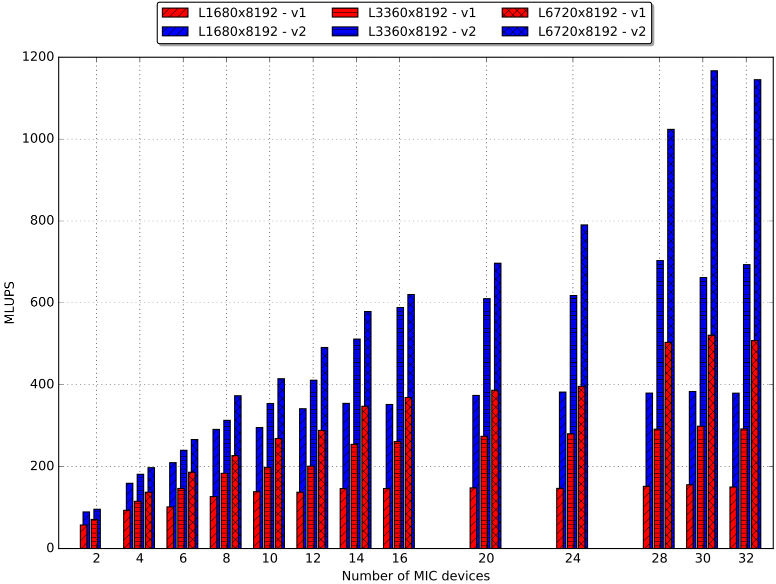
Outline

1. Lattice Boltzmann Method
2. Programming heterogeneous architectures
3. Data Layout Optimization
4. Load Balancing
5. Performances & Results

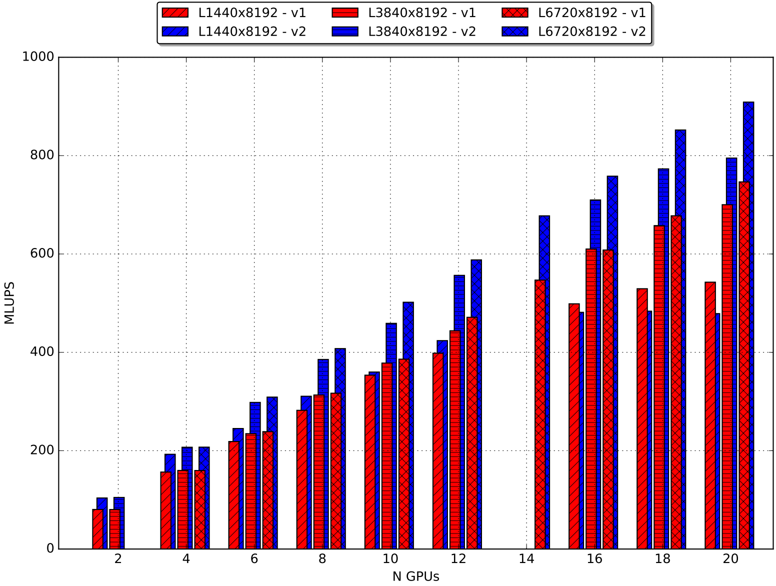
Tuning of cluster dimension



Scalability performances for Haswell + KNC



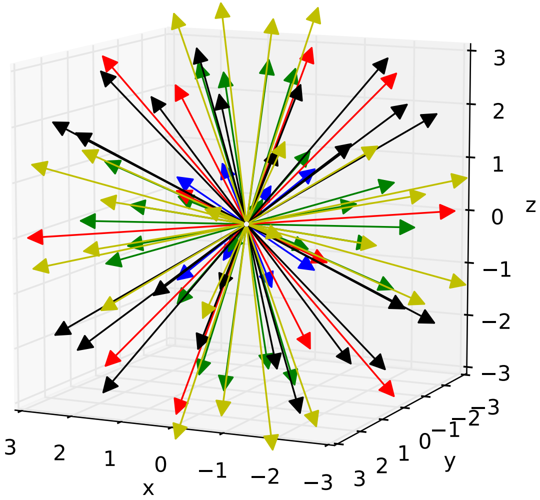
Scalability performances for Haswell + Tesla K80



Outline

1. Lattice Boltzmann Method
   - Lattice Boltzmann Equation
   - Computational Scheme
2. Programming heterogeneous architectures
   - Directive based programming models
   - Strategies for accelerator-based implementations
3. Data Layout Optimization
   - Array of Structures (AoS)
   - Structure of Arrays (SoA)
   - Clusterized Structure of Arrays (CSoA)
   - Clusterized Array of Structure of Arrays (CAoSoA)
4. Load Balancing
5. Performances & Results

Coming Next..



HPC-LEAP
EUROPEAN JOINT DOCTORATES

Heterogeneous implementation of the D2Q37 Lattice Boltzmann Method

Alessandro Gabbana

Università degli studi di Ferrara
Bergische Universität Wuppertal

September 27, 2016

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No' 642069