# Physics Simulations on multi- and many-core architectures

Sebastiano Fabio Schifano

University of Ferrara and INFN-Ferrara

SuperB: Computing R&D Workshop 2011

Ferrara

July 4-7, 2011

# Let me Introduce Myself

For those who do not know me:

- I'm not a physicist, but computer scientist
- I work with physicist since 1999
- I have been involved in several project to develop computing systems optimized for computational physics:
  - APEmille and apeNEXT: LQCD machines
  - AMchip: pattern matching processor, installed at CDF
  - Janus: 256 FPGA-based system
  - QPACE: TOP-GREEN 500 in Nov.'09 and July'10
  - AuroraScience

# Assessment of Multi- and Many-core Systems

More recently I have studied the performance of computing systems based on commodity multi- and many-core architectures for scientific computation.

## Systems

- multi-core: Cell-BE, Intel Nehalem and Westmere
- many-core: NVIDIA GPU Tesla C1060 and Fermi

## Applications

- Monte-Carlo simulations of Spin Glass systems
- Simulation of Rayleight-Taylor Instability: Lattice Boltzmann

# Assessment of Multi- and Many-core Systems

## Issues

- how to meet applications requirements and architecture features
- how to approach peak performance

## Methodology

- structured VS un-structured programs
- hardware-aware VS hardware-oblivious approach

## Goals

Study methodologies to program efficiently multi- and many-core systems.

# Simulation of the Rayleigh-Taylor (RT) Instability

Interface instability of two fluids of different densities triggered by gravity.

## Rayleigh-Taylor Instability

A cold-dense fluid over a less dense and warmer fluid triggers an instability that mixes the two fluid-regions (till equilibrium is reached).

E.g.: Pooring cold-wine over warmer water !!!

# The Lattice Boltzmann Method

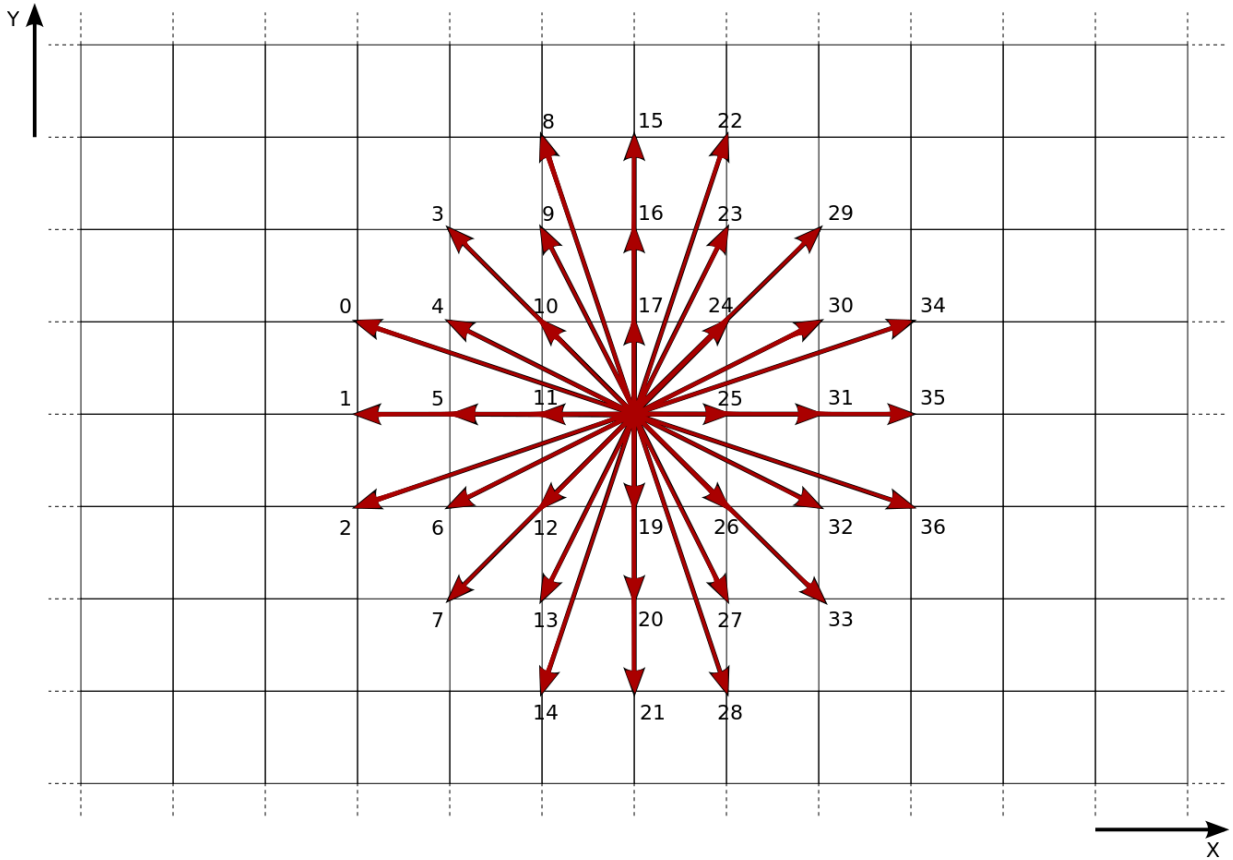
- Lattice Boltzmann methods (LBM) is a class of computational fluid dynamics (CFD) methods.
- Simulation of synthetic dynamics described by the discrete Boltzmann equation, instead of the Navier-Stokes equations.
- The key idea: **virtual particles** interacting by **streaming** and **collision** reproduce – after appropriate averaging – the dynamics of fluids.
- Easy to implement complex physics.
- **Good computational efficiency on MPAs**.

```
foreach time_step
  foreach lattice_point

    stream();

    collide();

  endfor
endfor
```

# The D2Q37 Lattice Boltzmann Model

- Correct treatment of:
  - Navier-Stokes equations of motion
  - heat transport equations
  - perfect gas state equation ($P = \rho T$)
- D2 model with 37 velocity components, 3D under development
- Suitable to study behaviour of **compressible** gas and fluids
- Optionally in the presence of **combustion** processes (chemical reactions turning cold-mixture of reactants into hot-mixture of burnt product)

# D2Q37: `stream()`



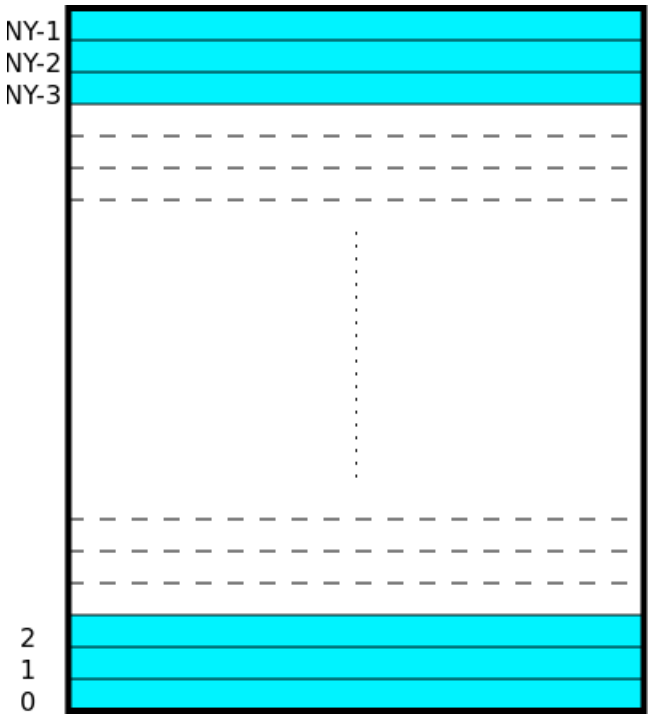

`stream()`:

- applies to each lattice-cell
- requires to access cells at distance 1,2, and 3.
- gathers populations at the edges of the arrows at the center point, that will be collided by next computational phase

# D2Q37: `bc()` and `collide()`

`bc()`: compute boundary conditions

adjusts values – e.g. set velocity to zero – at sites $y = 0 \ldots 2$ and $y = N_y - 3 \ldots N_y - 1$




`collide()`:

- computed for each lattice-cell
- computational intensive:
  requires $\approx$ 7820 DP ops.
- completely local:
  arithmetic operations require only the populations of the site

# D2Q37 CPU Implementation

```
typedef struct {
  double p[37];          // populations
  double u; double v;    // horizontal and vertical velocity
  double rho;            // density
  double temp;           // temperature
} pop_type;

for ( time_step = 0; time_step < MAXSTEP; time_step++ ) {
  stream();

  bc();

  collide;
}
```

- Each lattice-cell is represented by a struct variable of type `pop_type`
- At each point in the loop over time steps, each lattice-cell is processed by three main kernels:
  - `stream()`: moves particles among lattice-sites
  - `bc()`: interactions particles and up- and down-border
  - `collide()`: collision of particles

# D2Q37 CPU Implementation

Developed on the AuroraScience machine:

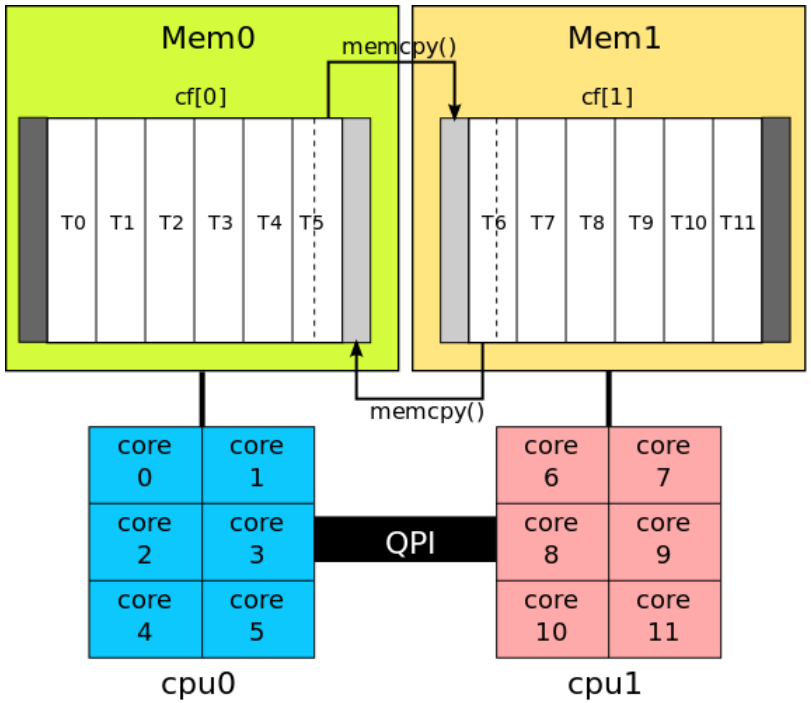
- node:
  - 2 six-core Intel Xeon X5660 CPUs per node
  - 12 MB L3-cache, 12 GB Ram
- system:
  - 32 nodes / chassis (16 front + 16 rear) 5 Tflops
  - 8 chassis / rack 40 Tflops
- network:
  - switched, based on standard Infiniband QDR adapters
  - point-to-point, based on a 3D-Torus network (FTNW$^*$) (latency $\approx 1$ $\mu$sec, bandwidth 1 GB/sec)
- *Symmetric Multi-Processor* (SMP) system:
  - programming view: single processor with 12 cores
  - memory address space shared among cores
- *Non Uniform Memory Access* (NUMA) system:
  memory access time depends on relative position of tread and data-structure

($*$) Developed by M. Pivanti, F. S. Schifano, and H. Simma

# D2Q37: Optimization on Multicore CPU System

Exploit parallelism at various levels:

- **node parallelism**: split the lattice over the nodes, each keeping a *sub-lattice*,
- **core parallelism**: split the sub-lattice over the cores of the node.
- **instruction parallelism**: process two lattice-sites in parallel exploiting vector SSE instructions within the core.

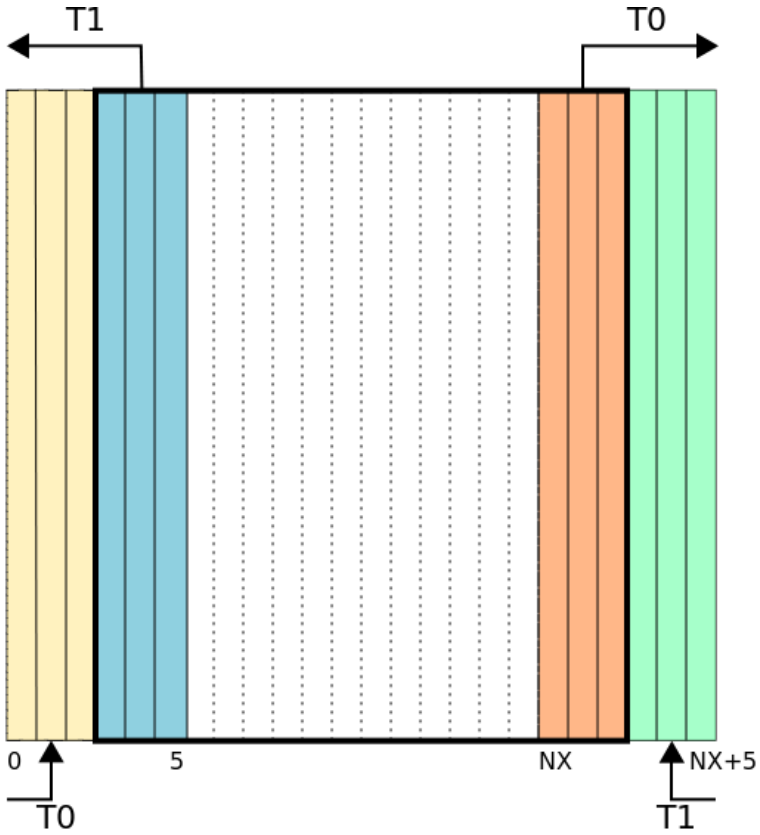



Optimize memory access:

- exploit cache-data reuse
- control memory allocation (NUMA)

# Node Parallelism

1. lattice size $L_x \times L_y$ is split over the nodes along X direction in sublattices of $\frac{L_x}{N_p} \times L_y$
2. on each node *borders* of neighbor sub-lattices are replicated
3. each sub-lattice is split over the cores




X-splitting make easy parallelization w/o bad impacts on performance.

# Core Parallelism

Each node is a SMP processor with 12 cores. Cores execute:

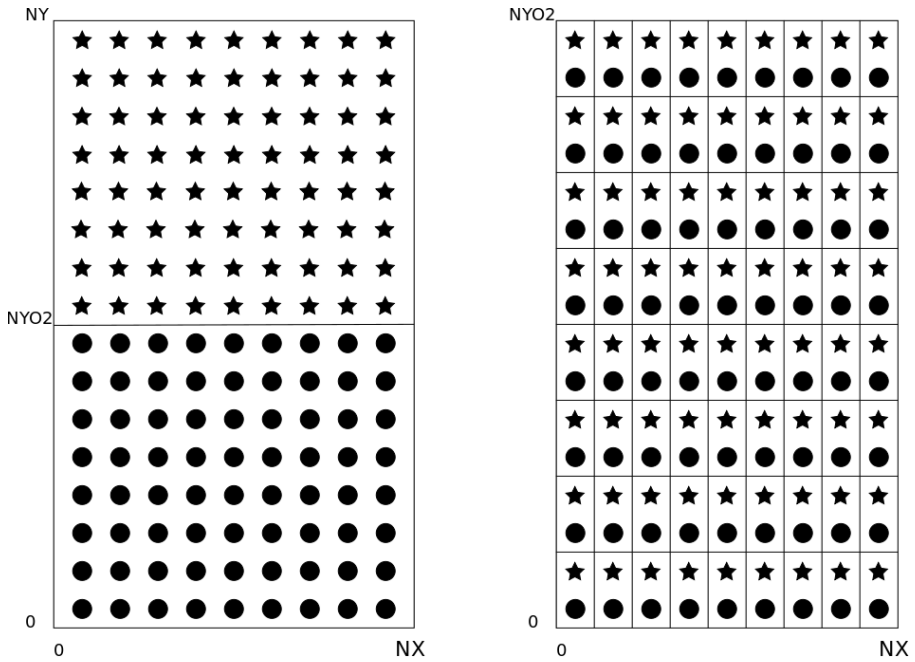
```
for ( step = 0; step < MAXSTEP; step++ ) {

  if ( tid == 0 || tid == 1 ) {
    comm();   // exchange borders
    stream(); // apply stream to left- and right-border
  } else {
    stream(); // apply stream to the inner part
  }

  pthread_barrier_wait(...);

  if ( tid == 0 )
    bc(); // apply bc() to the three upper row-cells

  if ( tid == 1 )
    bc(); // apply bc() to the three lower row-cells

  pthread_barrier_wait(...);

  collide();  // compute collide()

  pthread_barrier_wait(..);
}
```

Each node runs 12 threads, 1 for each available core.

# Instruction Parallelism

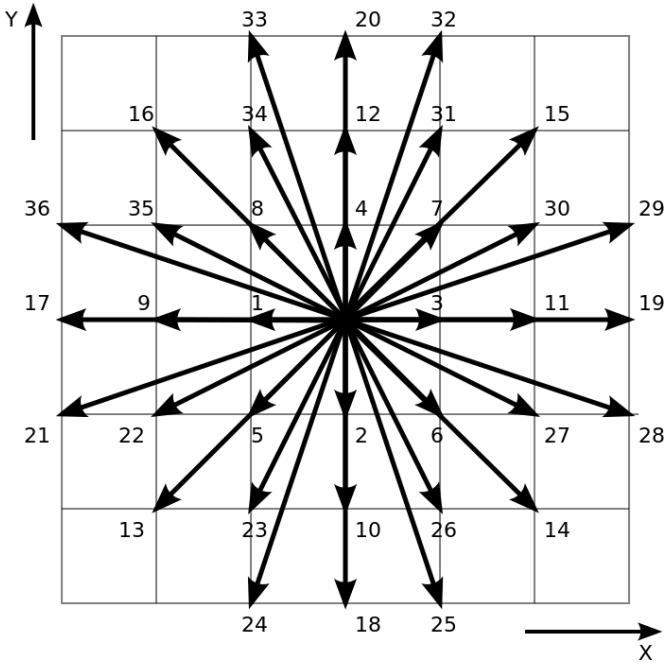
Components of two cells have been paired in a vector of two-doubles

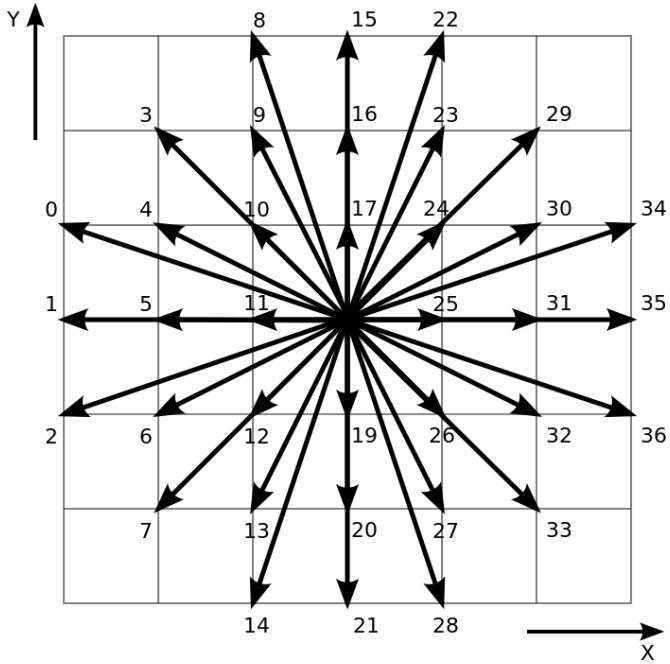
- we have used the gcc compiler
- each `v_pop_type` variable represent two cells
- two cells are processed in parallel




```
typedef double twoD __attribute__ ((vector_size(16)));

typedef struct {
  twoD p[37];       // populations
  twoD u; twoD v;   // horizontal and vertical velocity
  twoD rho;         // density
  twoD temp;        // temperature
} v_pop_type;
```

# Optimization of Stream




- Stream execution generates sparse memory accesses.
- Labelling of population is arbitrary. Cells are stored by-column.
- Reordering populations allow cache-reuse improving performance of stream (cache-aware).
- Use of **NUMA** library allows to control allocation of memory to avoid memory access conflicts.

# Optimization of Stream

| $L_x \times L_y$ | Size (GB) | Base | +Cache-reuse | +NUMA ctrl |
|---|---|---|---|---|
| $252 \times 8000$ | 1.4 | 0.18 | 0.12 (+33%) | 0.07 (+61%) |
| $480 \times 8000$ | 2.8 | 0.35 | 0.25 (+28%) | 0.13 (+62%) |
| $480 \times 16000$ | 5.4 | 0.72 | 0.52 (+27%) | 0.27 (+62%) |
| $480 \times 32000$ | 11.0 | 1.00 | 0.71 (+29%) | 0.54 (+46%) |

- Execution time (sec.) of stream for versions 1.x of the code on one node
- Numbers in brackets are the improvement w.r.t. the base version.
- Execution time grows proportionally with the size of data-set.
- NUMA reduces memory access conflicts and balances allocation.

# Performance Results (Versions 1.x)

| | Base | +Cache-reuse | +NUMA ctrl |
|---|---|---|---|
| $T_{pbc}$ | 0.34 s | 0.25 s | 0.12 s |
| $T_{stream}$ | 0.36 s | 0.26 s | 0.14 s |
| $T_{bc}$ | 0.9 ms | 0.5 ms | 0.2 ms |
| $T_{collide}$ | 0.39 s | 0.39 s | 0.39 s |
| $T_{time/site}$ | 12.5 ns | 11.2 ns | 8.7 ns |
| MLUps | 78 | 89 | 115 |
| $R_{max}$ | 23.8 % | 27.0 % | 35.2 % |

- Code versions 1.x running on 16 processing elements (192 cores).
- Grid size $L_x \times L_y = 4032 \times 16000$.
- $T_{time/site}$ is the time to process one lattice-cell.
- MLups, the number of grid sites updated per second.
- $R_{max}$ fraction of the peak (double precision).

# Merging Stream and Collide (Versions 2.x)

- If computation of combustion is not enabled, *stream* and *collide* phases can been merged together in a single-step (STEP 3).
- Optimizations for cache re-use and memory allocation, in the same way as described before.

| | Base | + Cache-reuse | + NUMA ctrl |
|---|---|---|---|
| STEP 1 | 0.06 s | 0.06 s | 0.06 s |
| STEP 2 | 1.36 ms | 1.32 ms | 0.64 ms |
| STEP 3 | 0.53 s | 0.47 s | 0.43 s |
| $T_{\text{time/site}}$ | 9.3 ns | 8.7 ns | 7.5 ns |
| MLUps | 103 | 113 | 130 |
| $R_{\text{max}}$ | 31.5 % | 34.4 % | 39.6 % |

- Code versions 2.x running on 16 processing elements (192 cores).
- Grid size $L_x \times L_y = 4032 \times 16000$.

# D2Q37 GPU Implementation

JUDGE - JÜlich Dedicated Gpu Environment

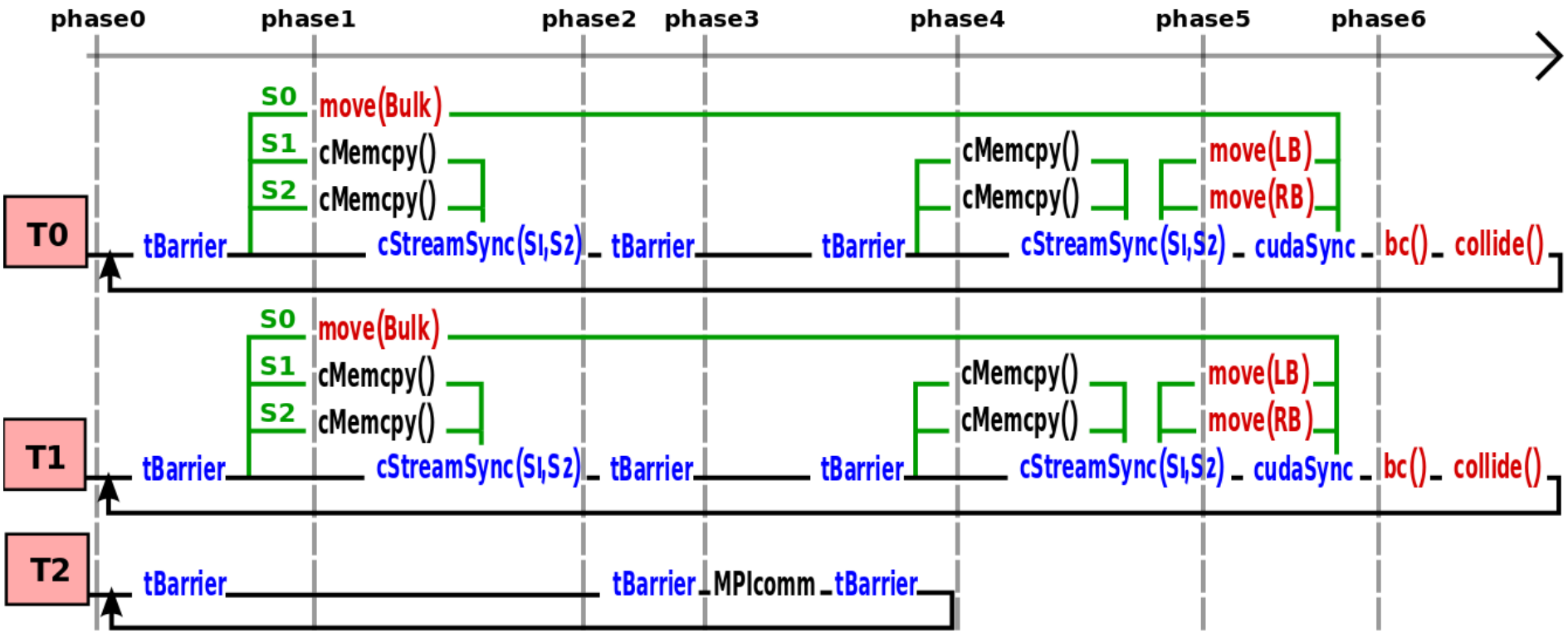
- **Compute Nodes**:
  - 54 Compute nodes IBM System x iDataPlex dx360 M3
  - node: 2 Intel Xeon X5650(Westmere) 6-core processor 2,66 GHz
  - Main memory: 96 GB
  - Network: IB QDR HBA
  - GPU: 2 NVIDIA Tesla M2050 (Fermi) 1,15 GHz (448 cores), 3 GB memory
- **Complete System**:
  - 648 cores
  - 108 graphic processors
  - 5,1 TB main memory
  - 62,5 Teraflops peak performance

# D2Q37 GPU Implementation

```
typedef struct {
  double p1 [NSITES]; // population  1 array
  double p2 [NSITES]; // population  2 array
  ...
  double p37[NSITES]; // population 37 array
} pop_type;

foreach ( timestep=0; timestep < MAX_STEP; timestep++ ) {
  comm ( );       // exchange Y borders
  move <<< grid, threads >>> (  );    // run stream
  bc <<< grid, threads >>> ( );       // run bc
  collide <<< grid, threads >>> ( ); // run collide
}
```

- SOA to exploit data-coalescing

# D2Q37 GPU Implementation




- two treads manage run on GPU
- one thread executes communication with neighbour nodes

# Performance Comparison: GPU vs CPU

| | GPU V1 | CPU V1 |
|---|---|---|
| comm | 0.20 ms | 10.00 ms |
| stream | 47.85 ms | 140.00 ms |
| bc | 0.60 ms | 0.20 ms |
| collide | 194.69 ms | 360.00 ms |
| GFLOps | 129.23 | 60.17 |
| $R_{max}$ | 25% | 38% |
| time/site | 0.06 $\mu$s | 0.13 $\mu$s |
| MLUps | 16.56 | 7.71 |

| | GPU V2 | CPU V2 |
|---|---|---|
| STEP 1 | 0.19 ms | 7.00 ms |
| STEP 2 | 1.18 ms | 0.64 ms |
| STEP 3 | 0.99 ms | 0.62 ms |
| STEP 4 | 193.45 ms | 410.00 ms |
| GFLOps | 160.59 | 72.41 |
| $R_{max}$ | 31% | 45% |
| time/site | 0.04 $\mu$s | 0.11 $\mu$s |
| MLUps | 20.59 | 9.28 |

- GPU-system: NVIDIA Fermi
- CPU-system: dual socket Intel six-core (Westmere, 160 Gflops DP peak)
- lattice size: $252 \times 16000$

# D2Q37 GPU: Scalability

| Strong Scalability Ver. V1 | | | | | | |
|---|---|---|---|---|---|---|
| #GPU | 2 | 4 | 8 | 16 | 32 | 64 |
| $P$ (Gflops) | 264 | 509 | 948 | 1504 | 2152 | 2138 |
| $S_r$ | 2 | 3.8 | 7.5 | 11.4 | 16.3 | 16.2 |

| Strong Scalability Ver. V2 | | | | | | |
|---|---|---|---|---|---|---|
| #GPU | 2 | 4 | 8 | 16 | 32 | 64 |
| $P$ (Gflops) | 322 | 644 | 1288 | 2336 | 2320 | 2329 |
| $S_r$ | 2 | 4 | 8 | 14.5 | 14.4 | 14.5 |

| Weak Scalability Ver. V1 | | | | | | |
|---|---|---|---|---|---|---|
| #GPU | 2 | 4 | 8 | 16 | 32 | 64 |
| $P$ (Gflops) | 262 | 520 | 1024 | 2088 | 4176 | 8320 |
| $S_r$ | 2 | 3.9 | 7.8 | 15.9 | 31.8 | 63.5 |

| Weak Scalability Ver. V2 | | | | | | |
|---|---|---|---|---|---|---|
| #GPU | 2 | 4 | 8 | 16 | 32 | 64 |
| $P$ (Gflops) | 326 | 652 | 1300 | 2603 | 5203 | 10385 |
| $S_r$ | 2 | 4 | 7.9 | 15.9 | 31.9 | 63.7 |

- strong scaling: lattice size $1024 \times 7168$
- weak scaling: each GPU allocate a lattice size $254 \times 14464$

# Intel MIC Systems

## Many Integrated Core Architecture


- Knights Ferry: development board
- Knights Corners: commercial product

### COmputing on Knights Architectures (COKA)

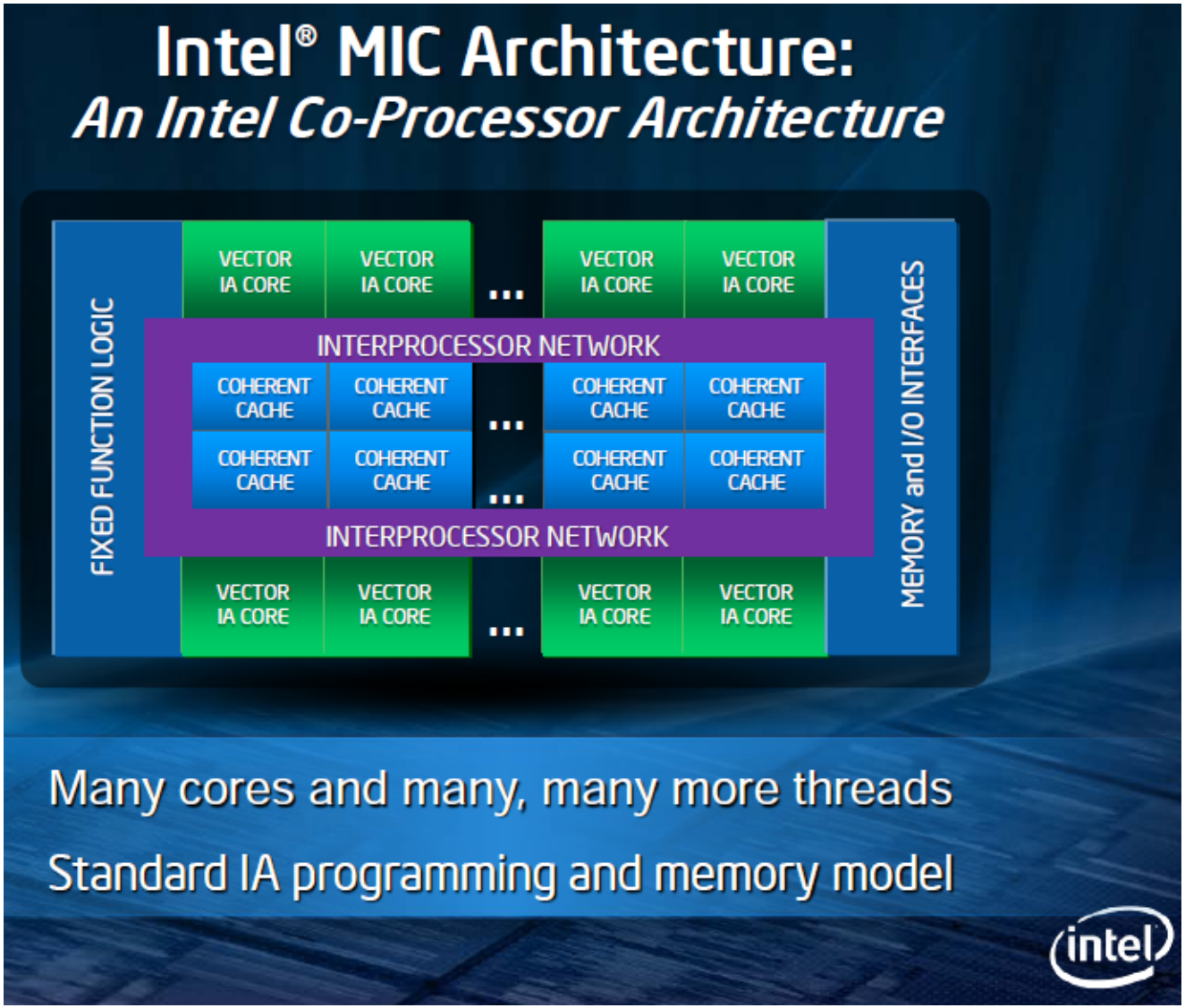
INFN-Ferrara, LNL and CNAF are starting the COKA project for studing performances of this class of architectures.

# Intel MIC Systems: Knights Ferry

- Yet another accelerator board
- PCIe interface
- Knights Ferry: 32 x86 core, 1.2 GHz
- each core has 32KB L1 instruction cache, 32KB L1 data cache, and 256KB L2 cache
- SSE unit: 16 SP, 8 DP
- multithreading: 4 threads / core
- 8 MB L3 shared coherent cache
- 1-2 GB GDDR5

Knights Corners could assemble up-to 64 cores.

# MIC Architectures



# Conclusions

What we have learnt:

- assessments of architecture features is important and crucial for application performances,
- use of multi- and many-core parallel systems require to exploit all possible levels of parallelism,
- in most cases a re-design of the program is necessary.

COKA project:

- study and compare performance of Knights architectures w.r.t. GPUs and CPUs,
- evaluate performance of applications relevant for INFN, both in the field of theoretical and experimental physics
- study programming methodology for many-core architectures to run at high efficiency.