Computing on Knights and Kepler Architectures

G Bortolotti, M Caberletti, G Crimi, A Ferraro, F Giacomini, M Manzali, G Maron,

M Pivanti, D Salomoni,S F Schifano, R Tripiccione, M Zanella

Sebastiano Fabio Schifano

University of Ferrara and INFN-Ferrara

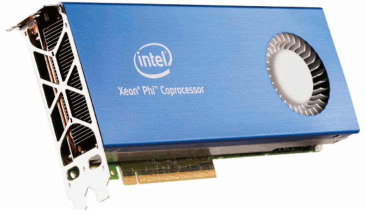
20th International Conference on Computing in

High Energy and Nuclear Physics

October 14-18, 2013

Amsterdam, The Netherlands

The emergence of accelerators



Use of accelerator based systems is today a common option for HPC.

Why are they interesting ?

	Xeon E5-2687	Tesla K20X	Xeon-Phi 7120P
#physical-cores	8	14 SMX	61
#logical-cores	16	2688	244
clock (GHz)	3.1	0.735	1.238
GFLOPS (DP/SP)	198.4/396.8	1.317/3.950	1.208/2.416
SIMD	AVX 64-bit	N/A	AVX2 512-bit
cache (MB)	20	1.5	30.5
#Mem. Channels	4	–	16
Max Memory (GB)	256	6	16
Mem BW (GB/s)	51.2	250	352
ECC	YES	YES	YES

- 1 Tflops in one device ✔
- nothing is for free ✘
  - manage high number of threads
  - exploit several levels of parallelism
  - hide latency host-device (Amdhal law)

The INFN COKA project

- originally **C**omputing **O**n **K**nights **A**rchitectures
- today **C**omputing **O**n **K**-**A**rchitectures to include also GP-GPUs

Architectures:

- “classic” multi-core
- Many-core: GPUs, Xeon-Phi (MIC)
- low-power systems

Goals:

- investigate performance of multi- and many-core processors
- assess programming methodologies

Focus

In the rest of the talk I focus only on benchmarking MIC-based systems using a LBM code.

LBM at glance

- Lattice Boltzmann method (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:
  - a set of **virtual particles** called **populations** arranged at edges of a discrete and regular grid
  - interacting by **propagation** and **collision** reproduce – after appropriate averaging – the dynamics of fluids.

- relevant features:
  - “Easy” to implement complex physics.
  - Good computational efficiency on MPAs.
  - Useful tool to investigate performances of processors.

The D2Q37 Lattice Boltzmann Model at Glance

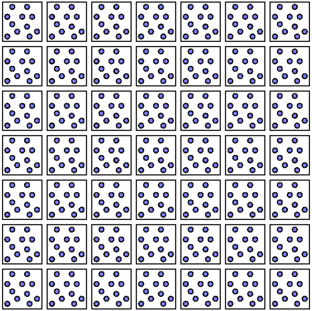
- Lattice Boltzmann method (LBM) is a class of computational fluid dynamics (CFD) methods
- simulation of synthetic dynamics described by the discrete **Boltzmann** equation, instead of the **Navier-Stokes** equations
- a set of **virtual particles** called **populations** arranged at edges of a discrete and regular grid
- interacting by **propagation** and **collision** reproduce – after appropriate averaging – the dynamics of fluids
- D2Q37 is a D2 model with 37 components of velocity (populations)
- suitable to study behaviour of **compressible** gas and fluids optionally in presence of **combustion** [1] effects
- correct treatment of *Navier-Stokes*, heat transport and perfect-gas ($P = \rho T$) equations

---

[1] chemical reactions turning cold-mixture of reactants into hot-mixture of burnt product.

Computational Scheme of LBM

```
foreach time-step

  foreach lattice-point
    propagate();
  endfor

  foreach lattice-point
    collide();
  endfor

endfor
```



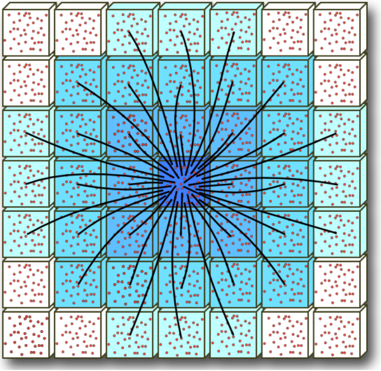
Embarassing parallelism

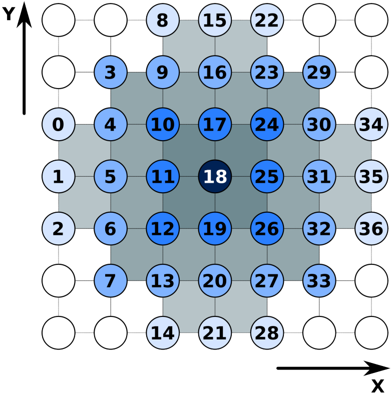
All sites can be processed in parallel applying in sequence propagate and collide.

Challenge

Design an efficient implementation to exploit a large fraction of available peak performance.

D2Q37: propagation kernel



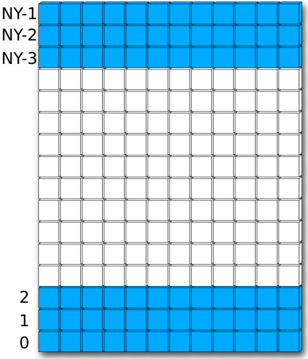


- require to access neighbours cells at distance 1,2, and 3
- generate memory-accesses with **sparse** addressing patterns

This kernel is strongly memory-bound.

D2Q37: boundary-conditions

- we simulate a 2D lattice with **period-boundaries** along *x*-direction
- at the top and the bottom boundary conditions are enforced:
  - to adjust some values at sites $y = 0 \dots 2$ and $y = N_y - 3 \dots N_y - 1$
  - e.g. set vertical velocity to zero



This step (bc) is computed before the collision step.

D2Q37: collision kernel

- collision is computed to each lattice-cell
- computational intensive: for the D2Q37 model, and requires $>$ **7600** DP operations
- completely local: arithmetic operations require only the populations associate to the site

This kernel is strongly compute-bound.

Optimizations relevant for Xeon-Phi performances

$P = f \times \#\text{cores} \times \text{NopPerCycle} \times \text{NflopPerOp}$

- **core parallelism**:
  the lattice is split among the 61 CPU-cores;
- **hyper-threading**:
  each core runs 2-4, threads to keep hardware pipelines busy and hide memory accesses latency;
- **vector programming**:
  each core process several sites in parallae data-set using vector (streaming) instructions (SIMD parallelism); in the case of Xeon-Phi up-to 8 double-precision values can be processed by each vector instructions.

Single-MIC implementation

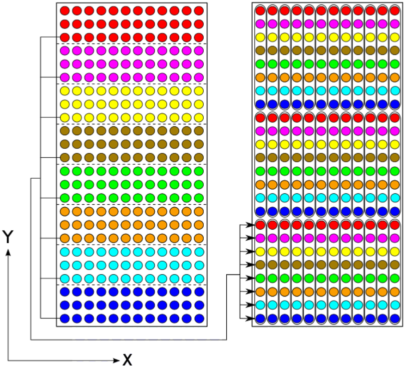
Each thread works on a sub-lattice and performs:

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

  if ( tid == 0 || tid == NTHR-1 ) {
    comm();       // exchange borders
    propagate(); // apply propagate to left- and right-border
  } else {
    propagate(); // apply propagate 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(..);
}
```

Offload a function that spawns several threads

Implementation: vector programming

Populations of 8 lattice-cells are packed in a AVX vector of 8-doubles



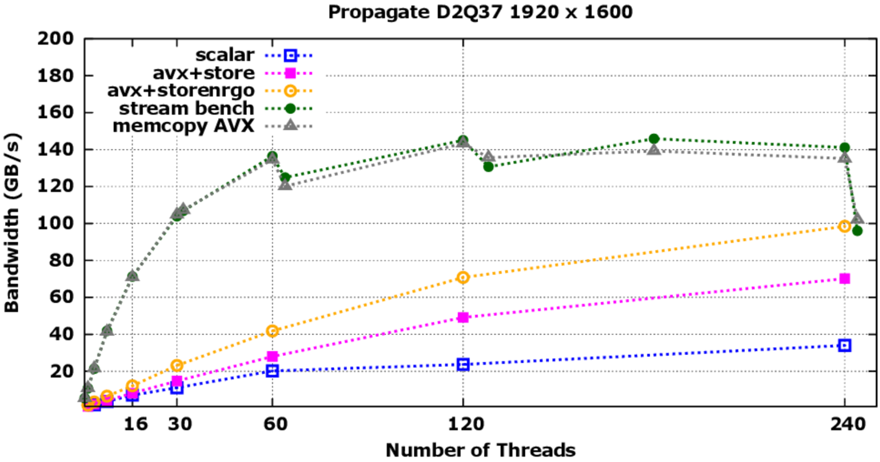
```
struct {
   __m512d vp0;
   __m512d vp1;
   __m512d vp2;
   ...
   __m512d vp36;
} vpop_t;

vpop_t lattice[LX][LY];
```

AoS scheme

Intrinsics

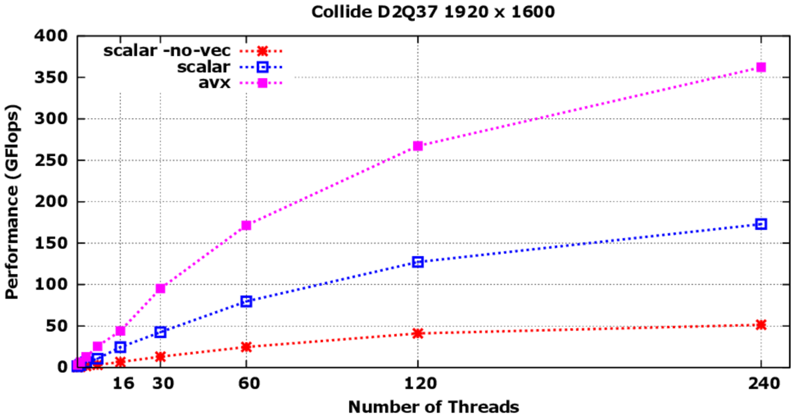
$d = a \times b + c \implies$ `d = _m512_fmadd_pd(a,b,c)`

Propagate



Performance are limited by internal ring bandwidth: ≈ 200 GB/s

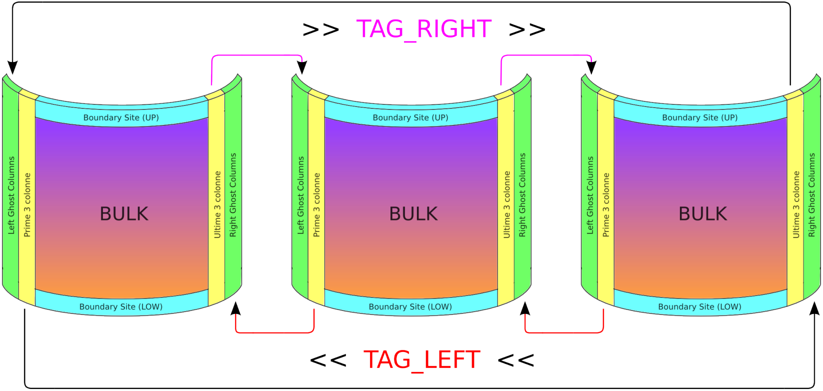
Collide



- scalar: `icc -mmic -O3 -openmp -novec`, 240 threads, $\epsilon \approx 5\%$
- scalar: `icc -mmic -O3 -openmp`, 240 threads, $\epsilon \approx 15\%$
- vector: intrinsic, openmp, 240 threads, $\epsilon \approx 30\%$

Single-host multi-MIC version

- partion lattice along X-direction among the MICs
- one MPI process per MIC
- MPI-process logically arranged in a ring



Single-host multi-MIC: implementation

Host offload execution of kernels

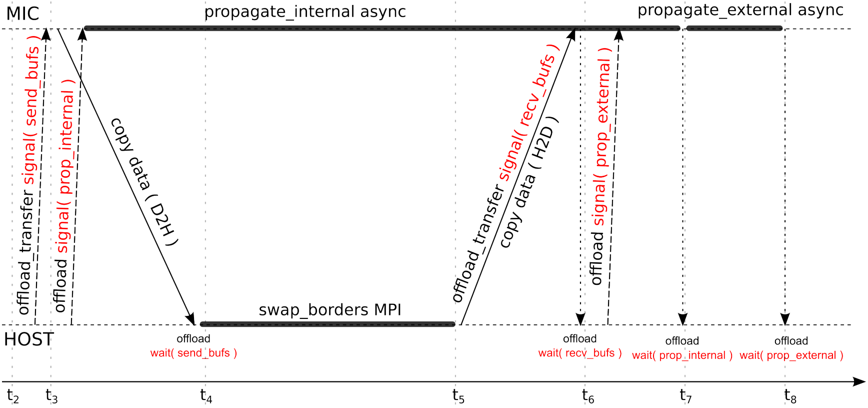
```
for ( step = 0; step < MAXSTEP; step++ ) {

  exchange_borders();

  #pragma offload target(mic:-1) { propagate(...) }

  #pragma offload target(mic:-1) { bc(...) }

  #pragma offload target(mic:-1) { collide(...) }

}
```

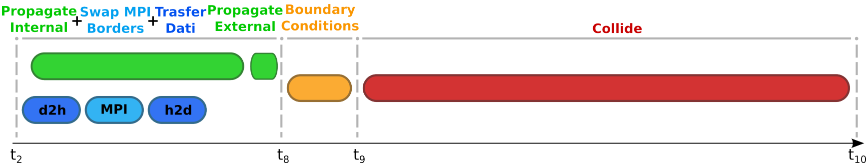
Single-host multi-MIC: exchange borders

1. copy the 3 right-most and left-most columns from device to host
2. exchange data with left and right neighbour
3. copy data from host to device

```
// transfer data from device to host (d2h)
#pragma offload_transfer: out( cf2[LEFT_THREE_COLS]  : REUSE into( send_L_buf ) )
#pragma offload_transfer: out( cf2[RIGHT_THREE_COLS] : REUSE into( send_R_buf ) )

// execute halos SWAP
MPI_Sendrecv(send_R_buf to mpi_rank_R, TAG_RIGHT, recv_L_buf to mpi_rank_L, TAG_RIGHT);
MPI_Sendrecv(send_L_buf to mpi_rank_L, TAG_LEFT, recv_R_buf to mpi_rank_R, TAG_LEFT);

// transfer data from host to device (h2d)
#pragma offload_transfer: in( recv_L_buf : REUSE into(cf2[LEFT_HALO] ))
#pragma offload_transfer: in( recv_R_buf : REUSE into(cf2[RIGHT_HALO]))
```

Overlapping Data-transfer & Computing





Overlapping Data-transfer & Computing

```
// launch asynchrouns transfer from device to host (d2h)
#pragma offload_transfer: out( cf2[LEFT_THREE_COLS] : REUSE into( send_L_buf ) )
  signal( &send_L_buf )
#pragma offload_transfer: out( cf2[RIGHT_THREE_COLS] : REUSE into( send_R_buf ) )
  signal( &send_R_buf )

// launch asynchronous execution of propagate kernel over BULK
#pragma offload: signal( &internal_prop_signal ){ propagate_m ( ... ); }

// wait end of d2h transfer
#pragma offload_wait: wait( &send_L_buf )
#pragma offload_wait: wait( &send_R_buf )

// execute halos SWAP
MPI_Sendrecv(send_R_buf to mpi_rank_R, TAG_RIGHT, recv_L_buf to mpi_rank_L, TAG_RIGHT);
MPI_Sendrecv(send_L_buf to mpi_rank_L, TAG_LEFT, recv_R_buf to mpi_rank_R, TAG_LEFT);

// launch asynchrous transfer from host to device (h2d)
#pragma offload_transfer: in( recv_L_buf : REUSE into(cf2[LEFT_HALO] )) signal( &recv_L_buf )
#pragma offload_transfer: in( recv_R_buf : REUSE into(cf2[RIGHT_HALO])) signal( &recv_R_buf )

// wait end of h2d transfer
#pragma offload_wait: wait( &recv_L_buf )
#pragma offload_wait: wait( &recv_R_buf )

// launch asynchronous execution of propagate over left- and right-columns
#pragma offload { propagate_m ( ... ); } signal(&external_prop_signal)

// wait end of propagate kernels
#pragma offload_wait: wait( &internal_prop_signal )
#pragma offload_wait: wait( &external_prop_signal )
```

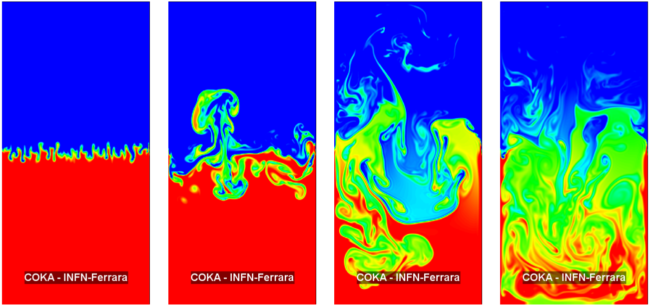
Results

#MIC	1	2	3	4
T_{prop} (msec)	164.3	86.6	62.1	51.0
T_{bc} (msec)	6.6	5.1	4.9	5.4
T_{col} (msec)	435.2	219.9	147.7	112.8
T_{tot} (msec)	606.1	311.9	215.1	169.4
Propagate (GB/s)	85	161	225	274
S_r	1.0X	1.90X	2.65X	3.22X
Collide (GFs)	358	709	1056	1383
S_r	1.0X	1.98X	2.95X	3.86X
Global P (GF/s)	257	500	725	920
MLUPS	38.93	72.63	109.68	139.23
S_r	1.0X	1.95X	2.82X	3.56X

- Single host with 4 MICs
- lattice 5760 × 4096
- collide: 6613 flop/site

Simulation of the Rayleigh-Taylor (RT) Instability

Instability at the interface of two fluids of different densities triggered by gravity.



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

Conclusion: performances comparison

Performance comparisons of our D2Q37 lattice boltzmann code on several platforms:

	Nvidia C2050	Intel dual E5-2680	Xeon-Phi 7120X	Nvidia K20X
propagate GB/s	84	60	98	155
ϵ	58%	70%	28%	62%
collide GF/s	205	220	362	565
ϵ	41%	63%	30%	43%
MLUPS	23	29	54	64
μJ / site	10.35	8.96	5.55	3.67

Performances of single-accelerators are a factor 2-3X better of a *classic* dual-processor CPU server.