Challenges in Programming Recent HPC Processors

(A Tribute to Lele Tripiccione)

Sebastiano Fabio Schifano

University of Ferrara and INFN

December 19, 2022
SM&FT 2022 Frontiers in Computational Physics
Bari - ITALY

Introduction

- several processors are available with different architectures
- application codes and perfomances can not be easily ported across different architectures
- compilers are still far from making it easy to move from one architecture to the other

How to use efficiently those processors ?

Optimization of Lattice-based (stencil) applications.

Lattice Boltzmann Methods - A real application as benchmark

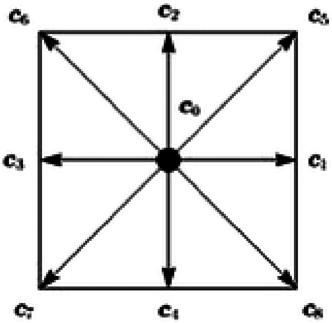
- a class of computational fluid dynamics (CFD) methods
- discrete **Boltzmann** equation instead of **Navier-Stokes** equations
- sets of **virtual particles**, called **populations**, arranged at edges of a D-dimensional $(D = 2, 3)$ lattice
- each population $f_i(x, t)$ has a given fixed lattice velocity $\boldsymbol{c}_i$, drifting – at each time step – towards a nearby lattice-site;
- populations evolve in discrete time according to the following equation:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{(eq)}\right)$$

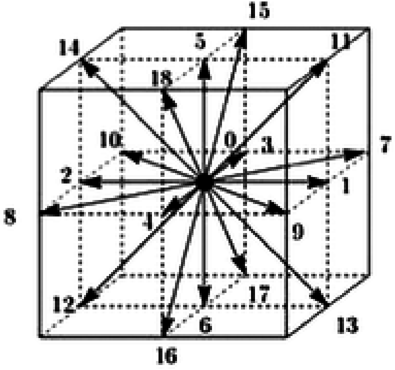
- macroscopic observables, like density ρ, velocity $\boldsymbol{u}$ and temperature T, are defined in terms of populations $f_i(x, t)$:

$$\rho = \sum_i f_i \quad \rho\boldsymbol{u} = \sum_i \boldsymbol{c}_i f_i \quad D\rho T = \sum_i \left|\boldsymbol{c}_i - \boldsymbol{u}\right|^2 f_i$$

LBM - A real application as benchmark



D2Q9



D3Q19

DnQk:

- *n* is the spatial dimension,
- *k* is the number of populations per lattice site

LBM Computational Scheme

Rewriting evolution equation as

$$f_i(\boldsymbol{y}, t + \Delta t) = f_i(\boldsymbol{y} - \boldsymbol{c}_i \Delta t, t) - \frac{\Delta t}{\tau} \left(f_i(\boldsymbol{y} - \boldsymbol{c}_i \Delta t, t) - f_i^{(eq)} \right)$$

being $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{c}_i \Delta t$, we can handle it by a two-step algorithm:

1. **propagate**:

   $$f_i(\boldsymbol{y} - \boldsymbol{c}_i \Delta t, t)$$

   gathering from neighboring sites the values of the fields f_i corresponding to populations drifting towards $\boldsymbol{y}$ with velocity $\boldsymbol{c}_i$;

2. **collide**:

   $$-\frac{\Delta t}{\tau} \left(f_i(\boldsymbol{y} - \boldsymbol{c}_i \Delta t, t) - f_i^{(eq)} \right)$$

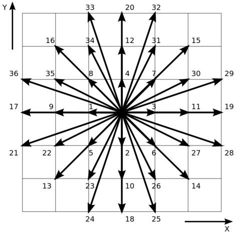
   compute the bulk properties density ρ and velocity $\boldsymbol{u}$, use these to compute the equilibrium distribution $f_i^{(eq)}$, and then relax the fluid distribution functions to the equilibrium state (τ relaxation time).

LBM Computational Scheme

```
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
- two relevant kernels:
  - *propagate* memory-intensive
  - *collide* compute-intensive
- *propagate* and *collide* can be fused in a single step;
- good tool to stress, test and benchmark computing systems.

D2Q37 LBM Application

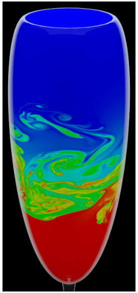


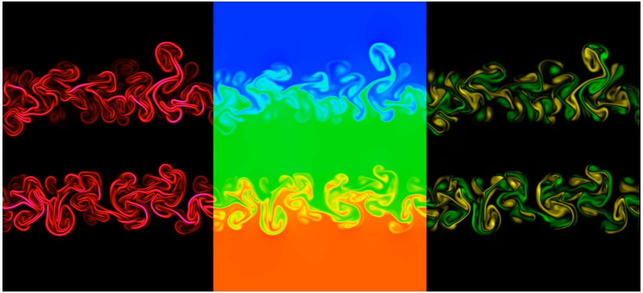
- D2Q37 is a 2D LBM model with 37 velocity components (populations);
- suitable to study behaviour of **compressible**gas and fluids optionally in presence of **combustion**[1] effects;
- include correct treatment of *Navier-Stokes*, heat transport and perfect-gas ($P = \rho T$) equations;
- used to study Rayleight-Taylor effects of stacked fluids at different temperature and density with periodic boundary conditions along one dimension;
- *propagate*: memory-intensive, access neighbours cells at distance 1,2, and 3, generate memory-accesses with **sparse** addressing patterns;
- *collide*: compute-intensive, require $\approx$ 6500 DP floating-point operations per lattice-site, it is local.

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

Rayleigh-Taylor Instability Simulation with D2Q37

Instability at the contact-surface of two fluids at different densities and temperatures triggered by the gravity.





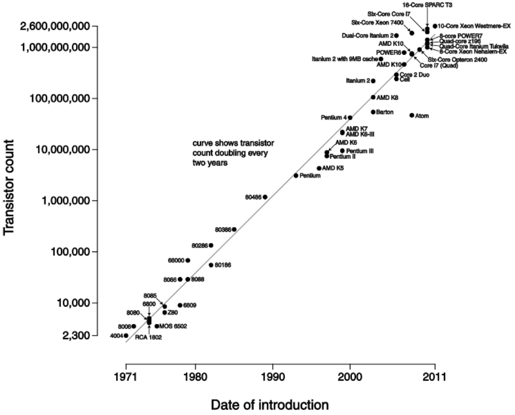
A cold-dense fluid over a less dense and warmer fluid triggers an instability that mixes the two fluid-regions (till equilibrium is reached).

Hardware Evolution

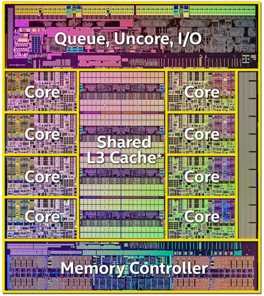
Multi-core architecture allows CPU performances to scale according to Moore's law.

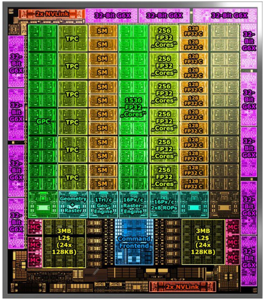
- increasing frequency beyond $\approx 3 - 4$ GHz is not possible ✘
- assembly more CPUs in a single silicon device ✔
- CPUs capable to execute vector instructions ✔
- great impact on application performance and design ✘
- move challenge to exploit high-performance computing from HW to SW ✘

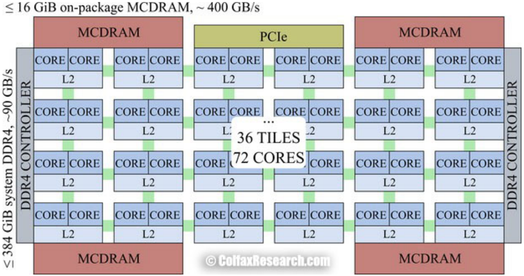
Microprocessor Transistor Counts 1971-2011 & Moore's Law



Typical multi- and many-core processors

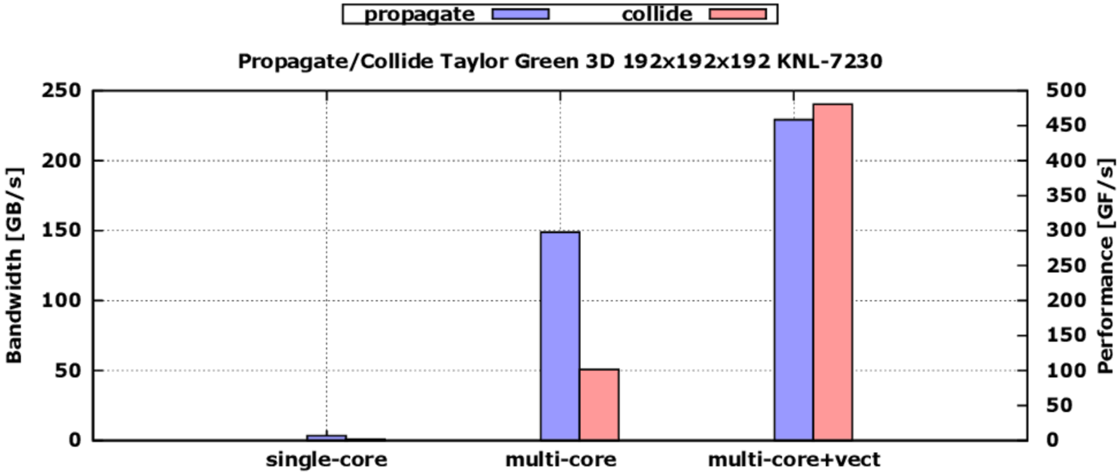




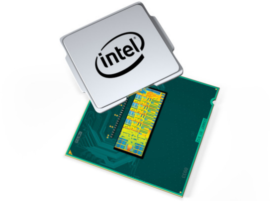


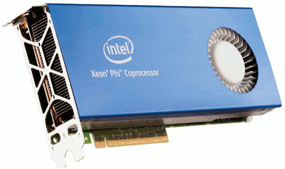
- several cores supporting each 2, 4 threads
- several hierarchy of caches (L1, L2, L3)
- large shared LL cache ($\mathcal{O}(10)$ MB for CPUs, **16 GB** for KNL)
- large VPUs (256-bit, 512-bit)
- **out-of-order** execution

Just to give an idea of what changes in practice !



Panorama of “modern” processors ... neglecting many details





Let’s compare new processors with the old APE CPU !

Panorama of “modern” processors ... neglecting many details

	apeNEXT	Xeon E5-2697 v4	Xeon 8160	Xeon Phy 7230	P100	V100	A100	H100
	Ape	Broadwell	Skylake	KNL	Pascal	Volta	Ampere	Hopper
Year	2002	2016	2017	2016	2016	2017	2021	2022
f [GHZ]	0.2	2.3	2.1	1.3	1.3	1.3	1.7	1.6
#cores / SMs	1	18	24	64	56	80	108	132
#threads / CUDA-cores	1	36	48	256	3584	5120	6912	16896
$\mathcal{P}_{DP}$ [GFlops]	1.6	650	1533	2662	4759	7000	9700	30000
β_{mem} [GB/s]	3.2	76.8	119.21	400	732	900	2039	3072
β_{net} [GB/s]	1.2	12.5	12.5	12.5	12.5	12.5	12.5	12.5
Watt	4	145	150	215	250	250	400	350
$\mathcal{P}/W$	0.4	4.5	10	12	19	28	24	86
$\mathcal{P}/\beta_{mem}$	0.5	8.5	13	6.6	6.5	7.7	4.7	9.8
$\mathcal{P}/\beta_{net}$	1.3	52	123	213	381	560	776	2400
$\mathcal{P} \times \lambda$	2.4e2	3.3e5	7.7e5	1.3e6	2.4e6	3.5e6	4.8e6	1.5e7

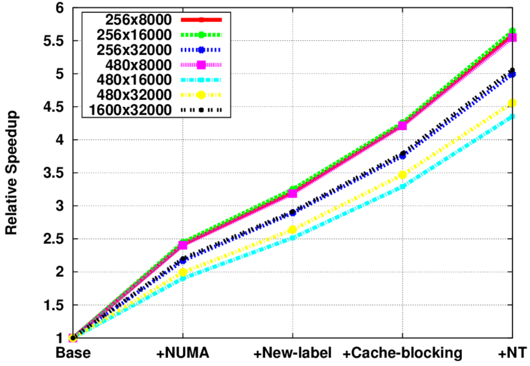
- several levels of parallelism: $\mathcal{P} = f \times \#\text{cores} \times \#\text{opPerCycle} \times \#\text{flopPerOp}$
- memory layout plays an important role also for computing performances.

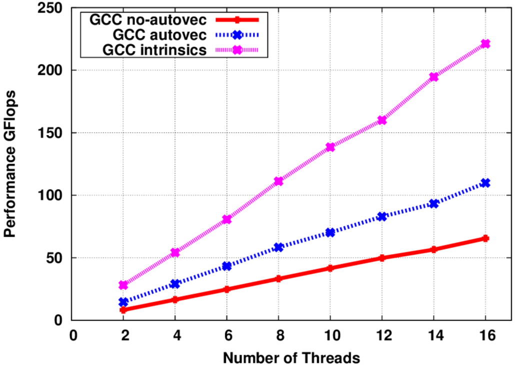
Programming Issues

$P = f \times \#\text{cores} \times \text{nInstrPerCycle} \times \text{nFlopPerInstr}$

- **core parallelism**:
  keep busy all available cores of a processor;
- **hyper-threading**:
  each core should run several (2, 3, 4) threads to keep busy hardware pipelines and hide memory access latency;
- **vectorization**:
  each core should process several data-elements (4, 8,. . .) in parallel using vector (streaming) instructions (SIMD parallelism);
- **thread and memory affinity**:
  many systems are dual-processors, threads of same process should be tied to one processor and access the closest memory bank;
- **cache-awareness**:
  exploit cache locality, cache reuse, avoid *reads for ownerships* (RFO). RFO is a read operation issued with intent to write to that memory address;
- **data layout**:
  relevants to expoit efficient vectorization and memory accesses.

Propagate and Collide Performance(dual-socket Xeon E5-2680 Sandybridge)

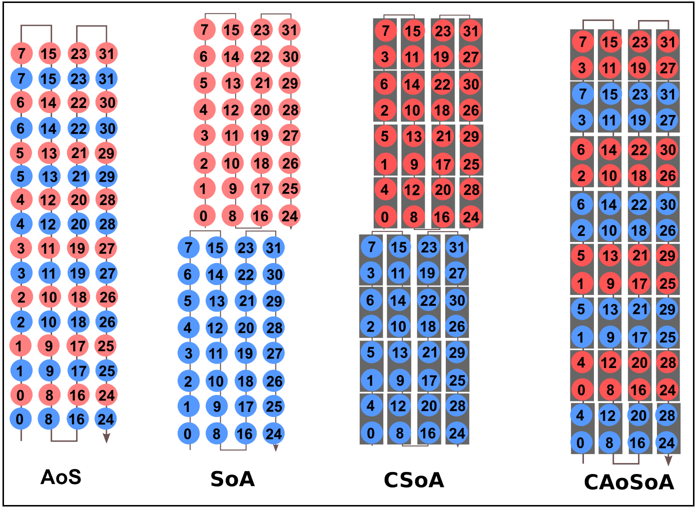




Including all optimizations: ≈ 58 MB/s, ≈ 68% of peak (85.3 MB/s), very close to memory-copy throughput (68.5 MB/s).

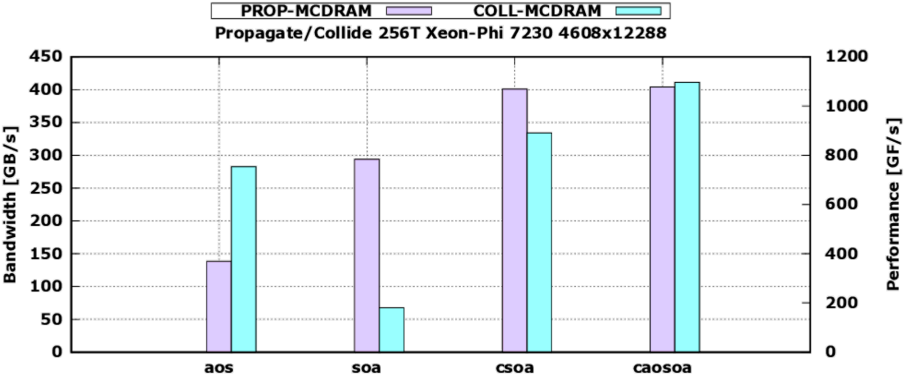
Impact of vectorization on a dual-socket Xeon E5-2680 Sandybridge: GCC no-autovec +18% of peak, GCC autovec +31% of peak, GCC intrinsics +62% of peak

Memory Data Layouts for LBM



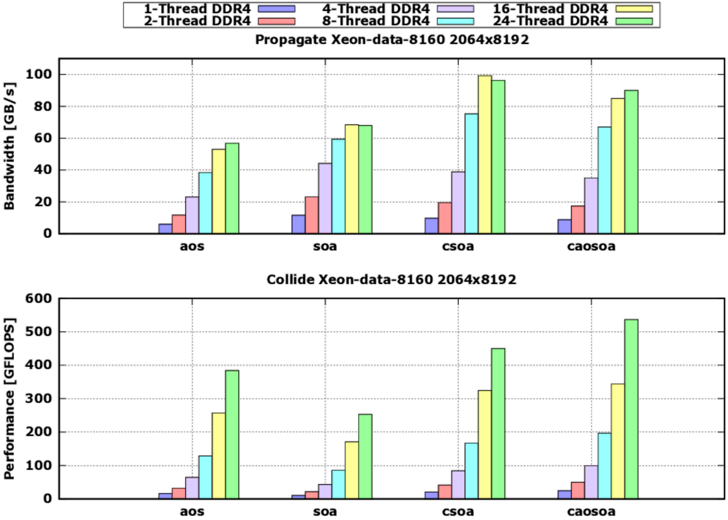
Sample Lattice 4 × 8 with two (blu and red) population per site.

Results: Propagate and Collide Performance on KNL



- Propagate:
  - FLAT-MCDRAM: performances increases from *AoS* → *SoA* → *CSoA*
  - *CSoA*: sustained bandwidth ≈ 400 GB/s (≈ 88% of the raw peak)
- Collide:
  - FLAT-MCDRAM: performance increases from *AoS* → *CSoA* → *CAoSoA*
  - *CAoSoA*: sustained performance of 1 Tflops (≈ 30% of raw peak)

Results: Performance on SkyLake



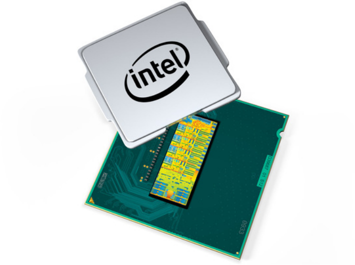
propagate ≈ 100 GB/s ≈ 85% of raw peak, *collide* ≈ 530 GFlops ≈ 35% of raw peak.

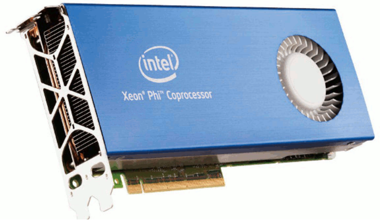
Results: VTUNE Analysis on KNL

Metric	*AoS*	*SoA*	*CSoA*	*CAoSoA*	Threshold
			propagate		
L2 Miss Rate	0.50	0.10	0.05	0.00	<0.20
L2 Miss Penalty	0.38	0.10	0.08	0.00	<0.05
L2 TLB Miss Ratio	0.00%	0.15%	0.00%	0.00%	<0.1%
L2 TLB Miss overhead	0.00	0.00	0.00	0.00	<0.05
			collide		
L2 Miss Rate	0.04	0.06	0.05	0.01	<0.20
L2 Miss Penalty	0.00	0.01	2.10	0.00	<0.05
L2 TLB Miss Ratio	0.00%	1.35%	0.96%	0.00%	<0.1%
L2 TLB Miss overhead	0.00	0.21	1.00	0.00	<0.05

- Threshold values suggested by the Intel VTUNE profiler
- `L2 CACHE Miss Rate`: fraction of memory references not found in `L2 CACHE`
- `L2 TLB Miss Overhead`: fraction of CPU cycles spent for data *page walks*

GP-GPUs: why are they interesting ?



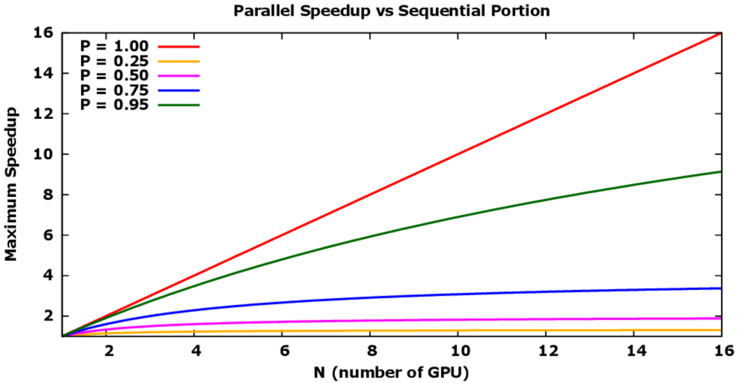


	Xeon E5-2697 v4	Xeon 8160	Xeon Phy 7230	P100	V100	A100
Year	2016	2017	2016	2016	2017	2020
$\mathcal{P}_{DP}$ [GFlops]	650	1533	2662	4759	7800	9700
β_{mem} [GB/s]	76.8	119.21	400	732	900	1555
$\mathcal{P}/W$	4.5	10	12	19	22	24

Why may not be interesting ?

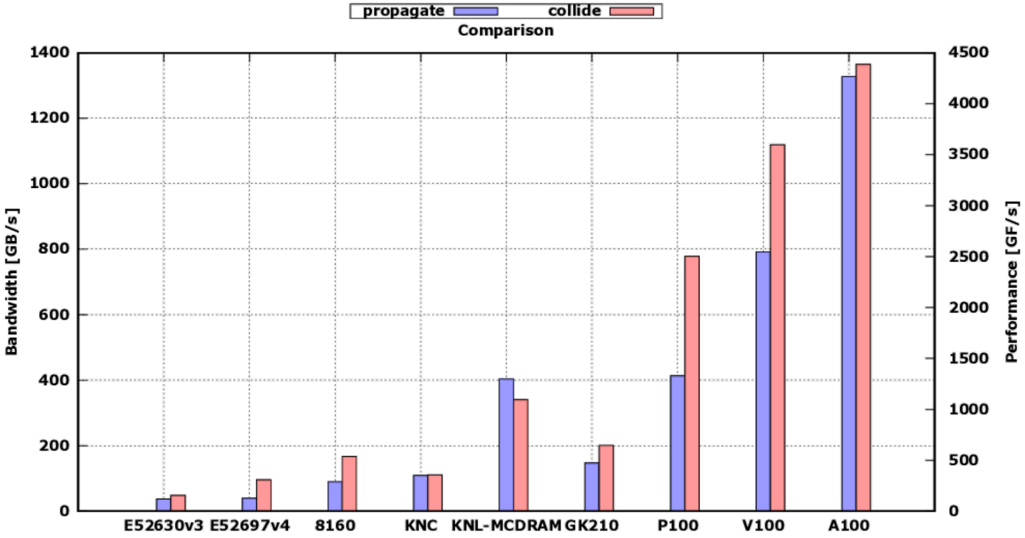
Amdahl's Law

Speedup of an accelerated program is limited by the fraction of time run on the host.

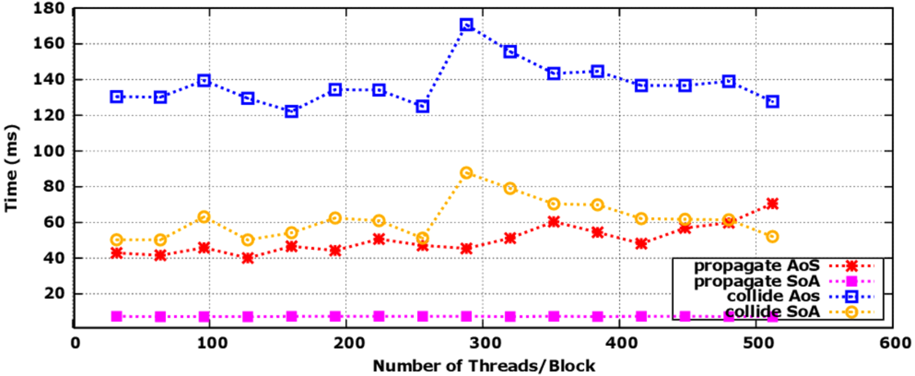


Accelerating the 3/4 of the code, the maximum theoretical achievable speedup is limited to 4 !!!

Why GPUs can be interesting: LBM Performance Comparison

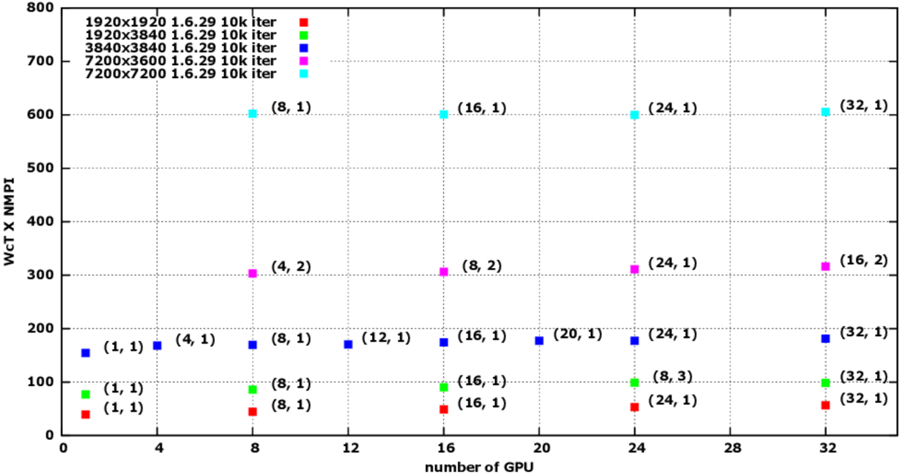


Memory layout for GPUs: AoS vs SoA



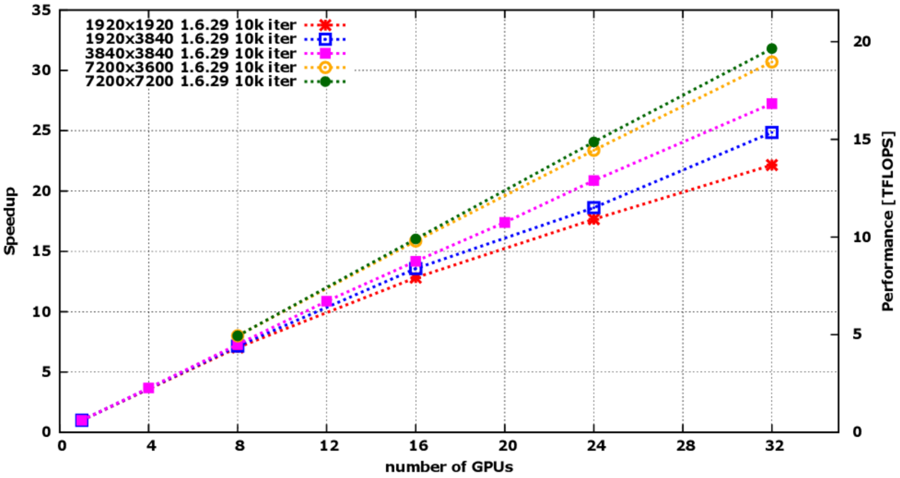
- Using SoA propagate is ≈ 10X faster, collide ≈ 2X faster
- SoA gives the best results because enable vector processing and coalescing access to memory.

Performance Scalability



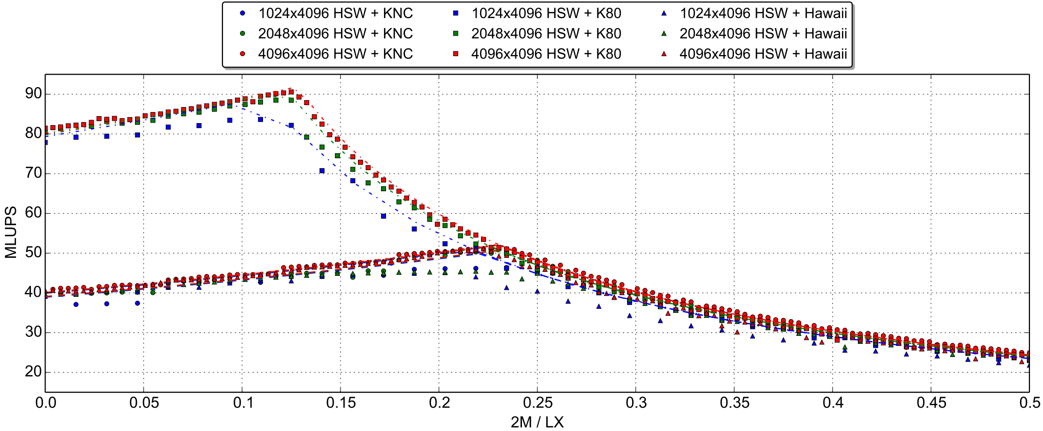
- in most cases the lowest time corresponds to 1D-tiling
- scalability is limited to a relatively small number of GPUs

Strong Scaling and Aggregate Performance



NVIDIA PSG cluster: 2X Haswell E5-2698 v3 CPUs at 2.30GHz each attached to two NVIDIA K80.

Real Heterogeneous Computing



Performance as a function of the fraction of lattice sites allocated on the host CPU.

- 1 MPI process per processor socket
- code running on CPU uses OpenMP and on GPU uses OpenACC
- same code running on HSW+K80, HSW+KNC, HSW+Hawaii

FPGA as accelerators ?

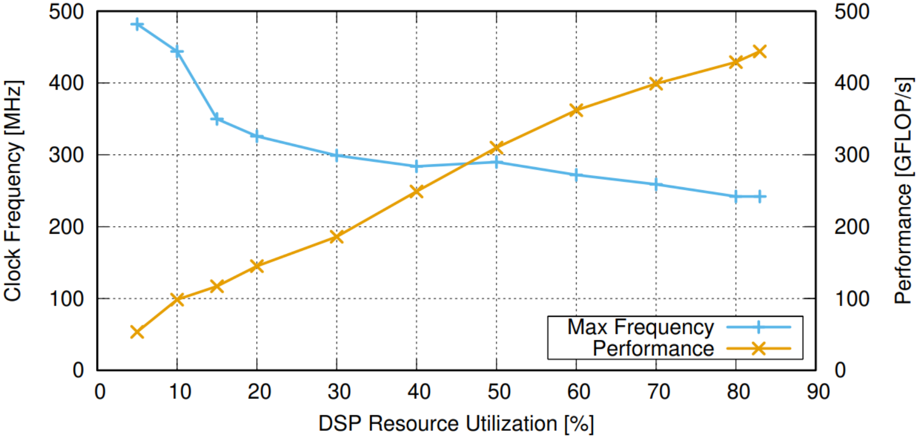
- latest generation of FPGA board can be used as accelerators, e.g. Xilinx Alveo
- peak performance and bandwidth is high
- programmed using high-level frameworks, e.g. HLS

Board	Frequency	Performance	URAM Bandwidth	HBM Bandwidth	RAM Bandwidth
U280	300 MHz	≈ 400 GFLOP/s	≈ 3.7 TB/s	≈ 460 GB/s	≈ 38 GB/s
U250	300 MHz	≈ 530 GFLOP/s	≈ 4.9 TB/s	–	≈ 76 GB/s
U50	300 MHz	≈ 260 GFLOP/s	≈ 2.5 TB/s	≈ 460 GB/s	–

FER benchmark

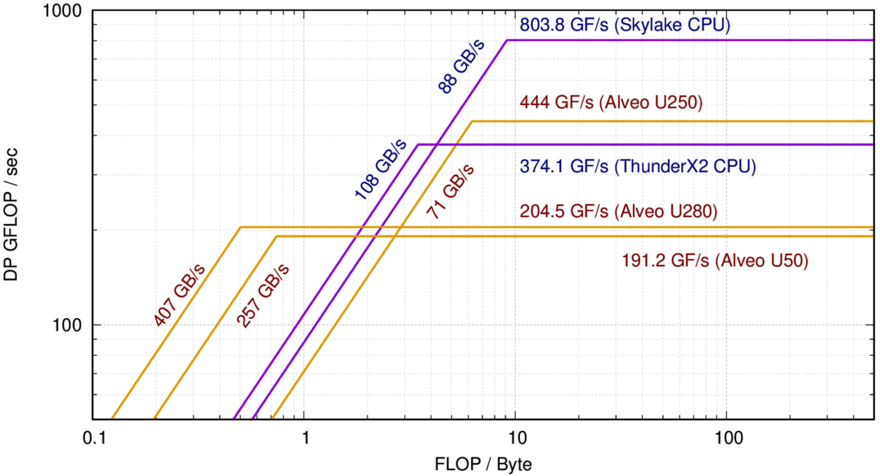
A tool to make Roofline plots for FPGA (UniFe + INFN).

FPGA as accelerators ?



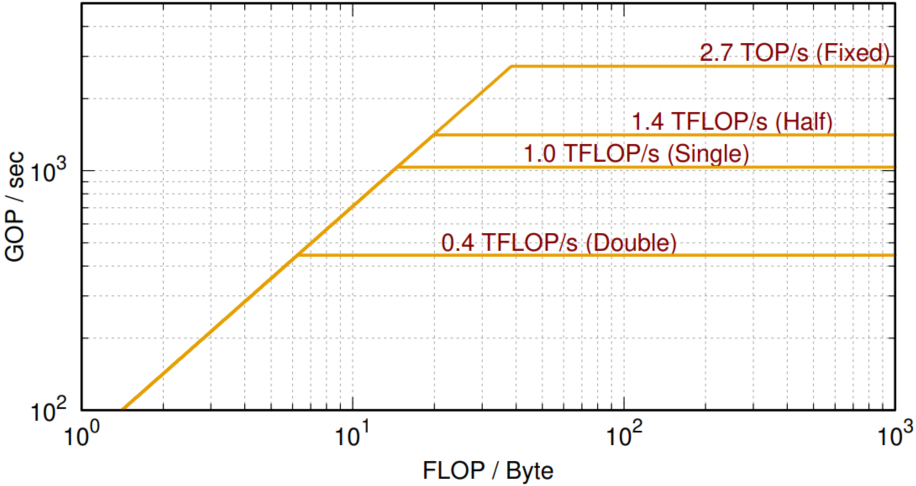
Maximum clock frequency (left axis) achieved synthesizing FER for the Alveo U250 and the corresponding DP-FP performance (right axis), as function of the DSP fraction used in the design.

FPGA as accelerators ?



Roofline plots comparison between U280/U250/U50 FPGAs and Intel Skylake CPU obtained by the ERT benchmark, and Marvell ThunderX2 CPU run using a version of ERT optimized by Arm.

FPGA as accelerators ?



More competitive performances can be achieved using lower precision operations.

Conclusions

Multi and many-core architectures have a big inpact on programming and development of codes.

- Efficient programming requires to exploit several features of hardware systems: core parallelism, data parallelism, cache optimizations, NUMA controls
- several architectures and several ways to program them
- data structure have a big impact on performance
- portability of code and performance is not yet fully supported

Programming commodity HPC system require deep knowledge ! For now no common approaches are available and it is necessary to deal with different programming frameworks.

Acknowledgments

- Raffaele Tripiccione (Lele), University of Ferrara and INFN
- Filippo Mantovani (BSC), Enrico Calore (INFN), Alessandro Gabbana (TUe)
- Marcello Pivanti, Fabio Pozzati, Alessio Bertazzo, Gianluca Crimi, Elisa Pellegrini, Nicola Demo, Giovanni Pagliarini, . . .

Results presented here have been developed in the frameworks of the INFN projects: COKA, COSA and SUMA. Thanks to CINECA, INFN, BSC and JSC for access to their systems.