D2Q37 LBM

Sebastiano Fabio Schifano

University of Ferrara and INFN

June 21-22, 2022
Jülich NVIDIA Lab Workshop

Jülich - Germany

Introduction

- several processors are available with different architectures
- application codes and perfomances can not be easily ported across different processor architectures
- compilers have improved but are far from making it easy to move from one architecture to the other
- find out how to program efficiently latest processors "minimizing" changes in the codes it is necessary

Focus

Optimization issues of Lattice-based (stencil) applications on latest multi- and many-core processors

Outline

Lattice Boltzmann Methods (LBM)

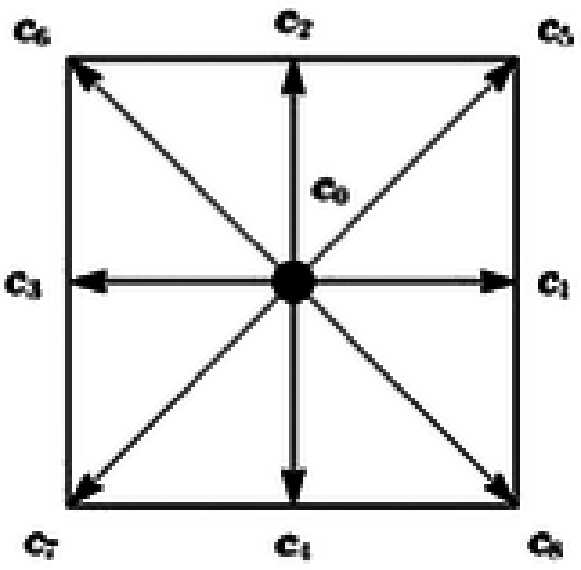
- a class of computational fluid dynamics (CFD) methods
- discrete **Boltzmann** equation instead of **Navier-Stokes** equations
- sets of **virtual particles**, called **populations**, arranged at edges of a D-dimensional ($D = 2, 3$) lattice
- each population $f_i(x, t)$ has a given fixed velocity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

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

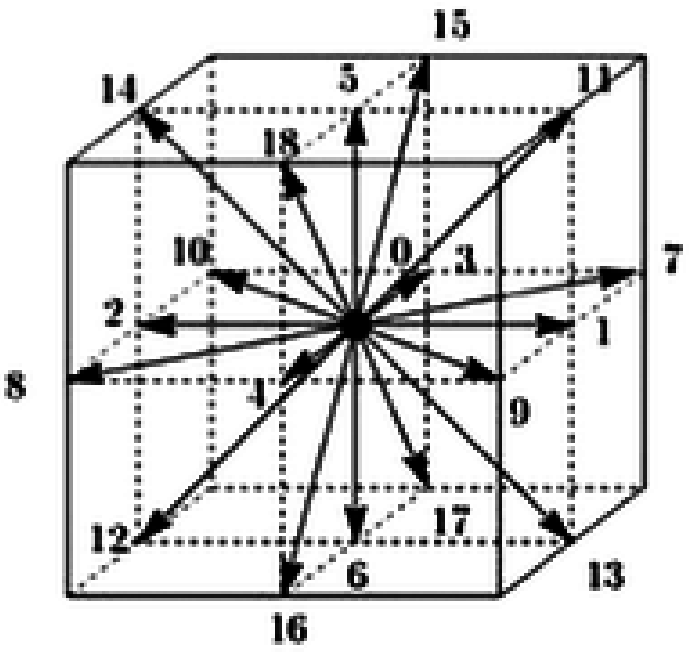
- macroscopic observables, like density ρ, velocity $\boldsymbol{u}$ and temperature T, are defined in terms of populations $f_i(x, t)$:

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

Different LBM models



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 ,may define 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).

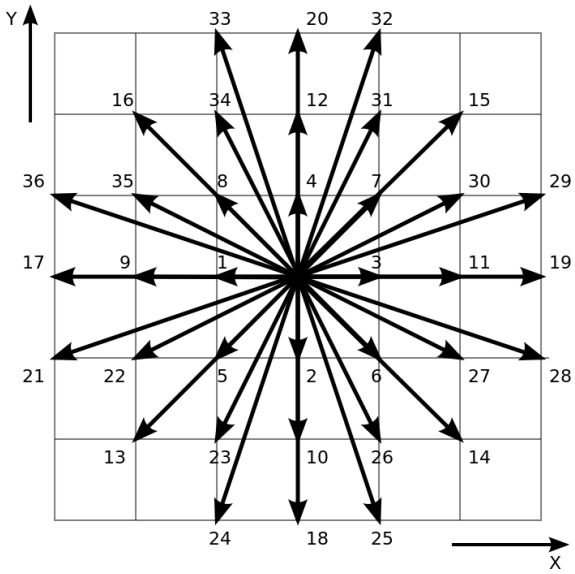
LBM Computational Scheme

```
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* may be fused in most cases into a single step

Good and easy tool to stress, test and benchmark computing systems.

D2Q37 LBM Application

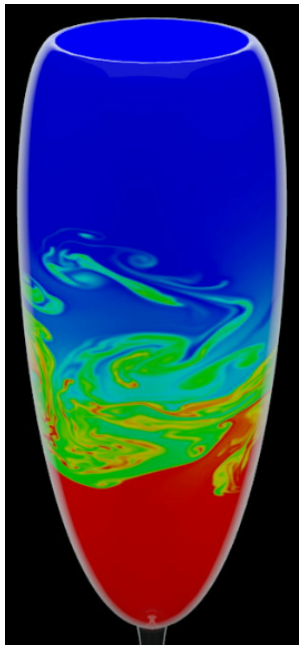


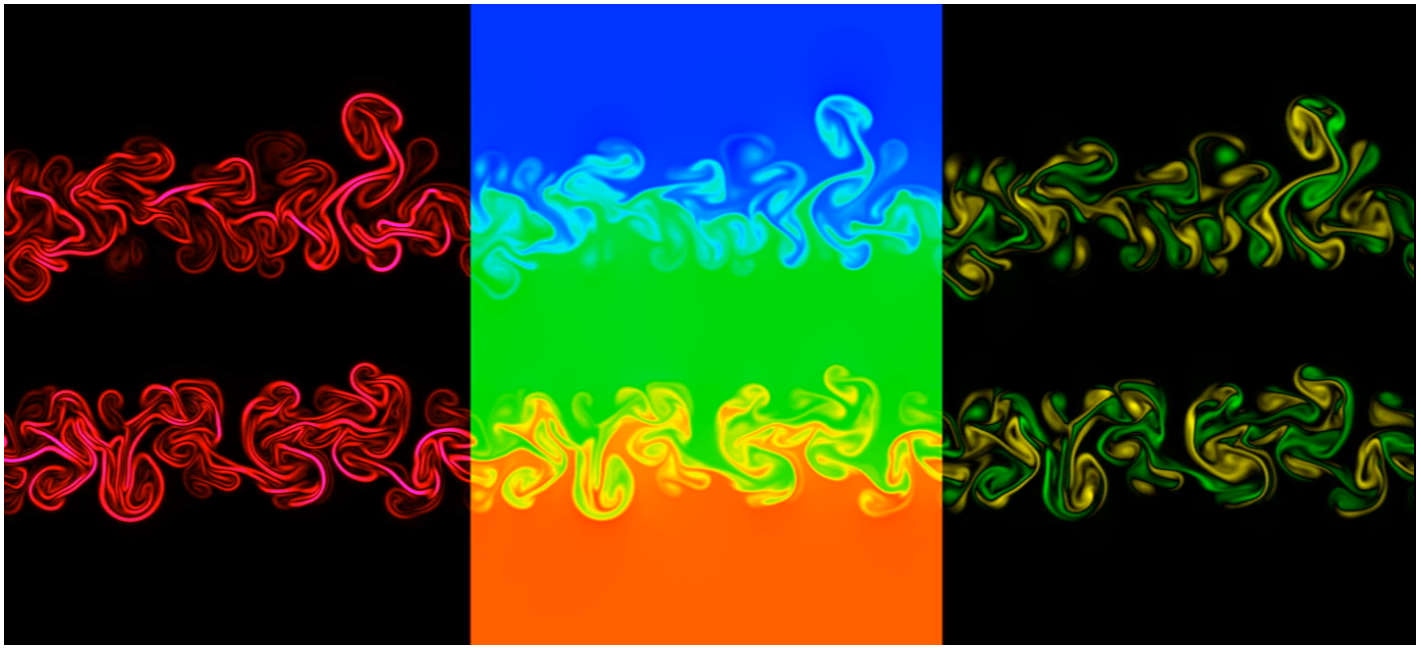
- D2Q37 is a 2D LBM model with 37 velocity components (populations);
- suitable to study behaviour of **compressible** gas and fluids optionally in presence of **combustion**[1] effects;
- include correct treatment of *Navier-Stokes*, heat transport and perfect-gas ($P = \rho T$) equations;
- used to 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, requires $\approx$ 6500 DP floating-point operations, is local.

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

Rayleigh-Taylor Instability Simulation with D2Q37

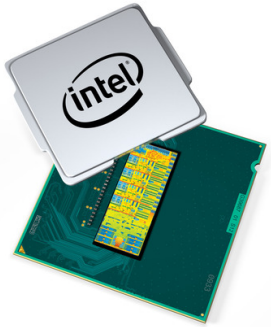
Instability at the contact-surface of two fluids of different densities and temperature triggered by gravity.

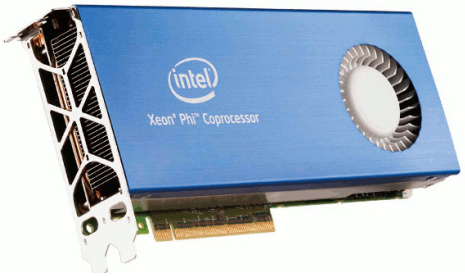




A cold-dense fluid over a less dense and warmer fluid triggers an instability that mixes the two fluid-regions (till equilibrium is reached).

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



	apeNEXT Ape	Xeon E5-2697 v4 Broadwell	Xeon 8160 Skylake	Xeon Phy 7230 KNL	P100 Pascal	V100 Volta
Year	2002	2016	2017	2016	2016	2017
f [GHZ]	0.2	2.3	2.1	1.3	1.3	1.3
#cores / SMs	1	18	24	64	56	80
#threads / CUDA-cores	1	36	48	256	3584	5120
$\mathcal{P}_{DP}$ [GFlops]	1.6	650	1533	2662	4759	7000
β_{mem} [GB/s]	3.2	76.8	119.21	400	732	900
β_{net} [GB/s]	1.2	12.5	12.5	12.5	12.5	12.5
Watt	4	145	150	215	250	250
$\mathcal{P}/W$	0.4	4.5	10	12	19	28
$\mathcal{P}/\beta_{mem}$	0.5	8.5	13	6.6	6.5	7.7
$\mathcal{P}/\beta_{net}$	1.3	52	123	213	381	560
$\mathcal{P} \times \lambda$	240	331’000	766’000	1’331’000	2’379’000	3’500’000

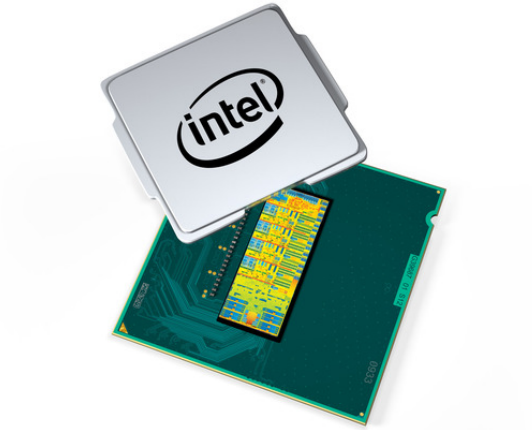
β_{net} for apeNEXT is 200 MB/s x 6, $\lambda = 150$ ns

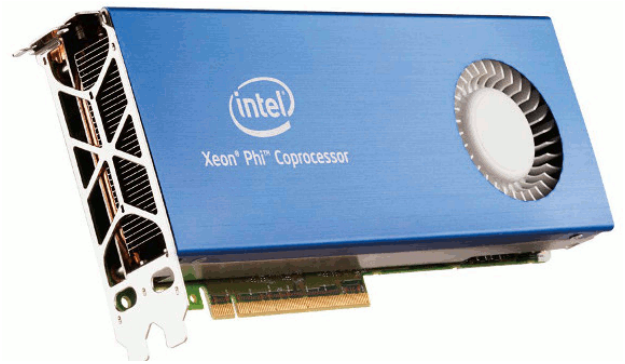
β_{net} for P100 and V100 is Mellanox 4X EDR $\approx$ 12.5 GB/s, $\lambda = 500$ ns

β_{net} for Xeon is OmniPath 100 Gbit/s = 12.5 GB/s, $\lambda = 500$ ns

- Several levels of parallelism
- Memory layout plays an important role also for computing performances.

GP-GPUs: why are they interesting ?





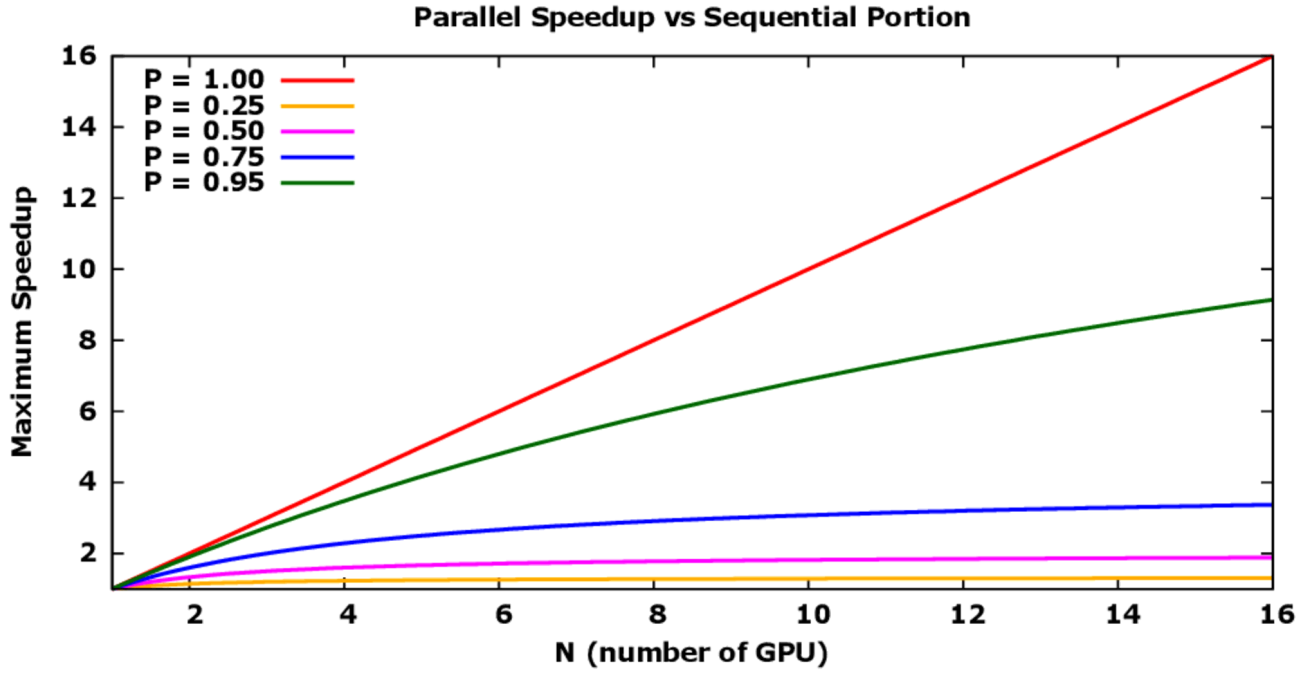
	Xeon E5-2697 v4	Xeon 8160	Xeon Phy 7230	P100	V100	A100
Year	2016	2017	2016	2016	2017	2020
$\mathcal{P}_{DP}$ [GFlops]	650	1533	2662	4759	7800	9700
β_{mem} [GB/s]	76.8	119.21	400	732	900	1555
$\mathcal{P}/W$	4.5	10	12	19	22	24

Which fraction of GPU peak performance can be effectively used ? How the code should be structured ?

Why may not be interesting ?

Amdahl's Law

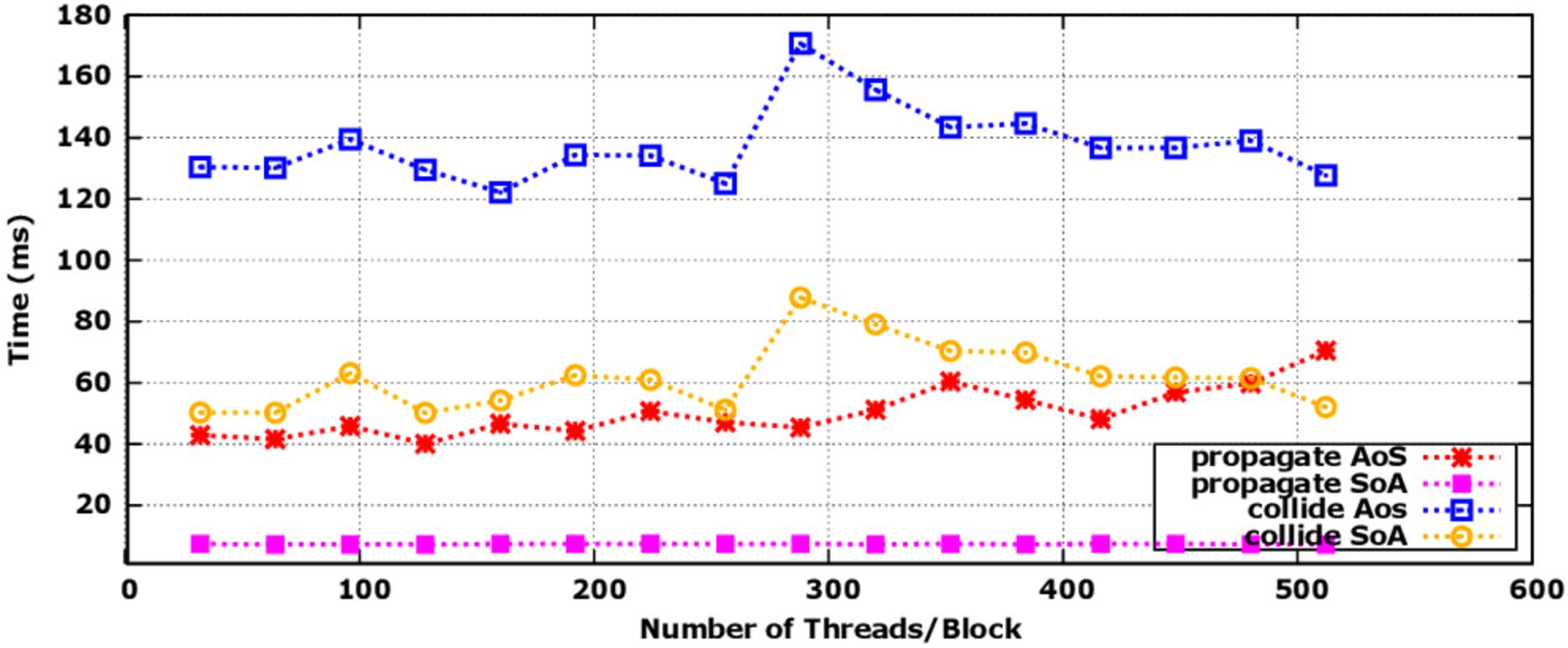
Speedup of an accelerated program is limited by the fraction of time run on the host.



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

Memory layout for GPUs: AoS vs SoA

On GPUs memory layout is relevant for performances:

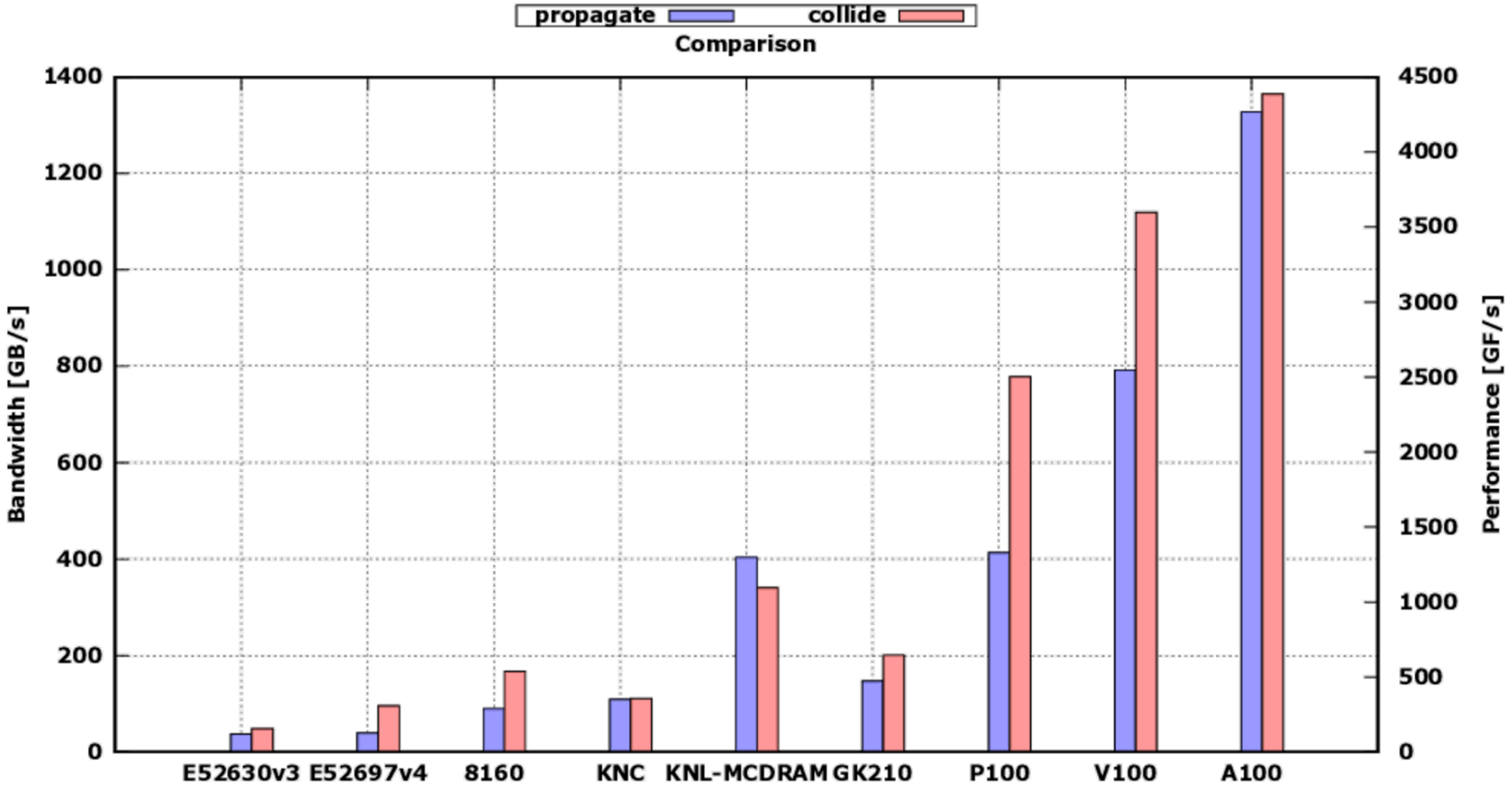


NVIDIA K40 GPU

- Using SoA propagate is $\approx$ 10X faster, collide $\approx$ 2X faster
- SoA enable vector processing and coalescing access to memory deliver better results.

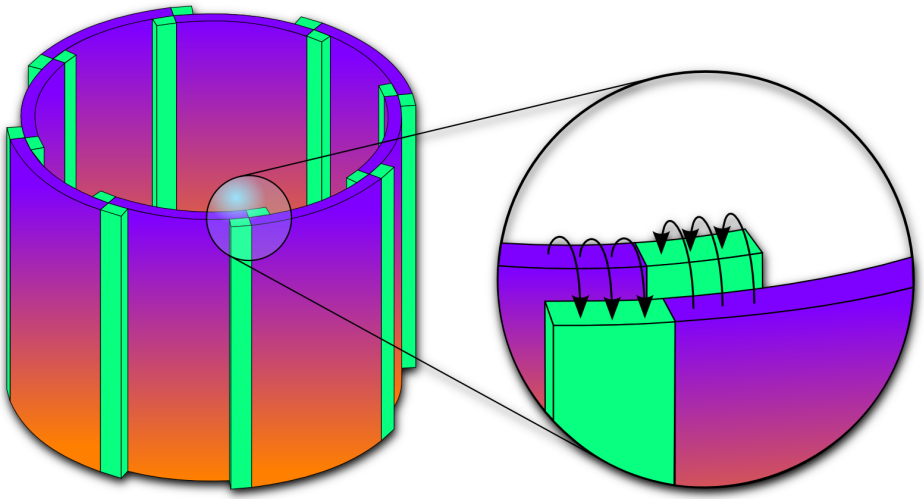
Why GPUs can be interesting

	Xeon E5-2697 v4	Xeon 8160	Xeon Phy 7230	P100	V100	A100
$\mathcal{P}_{DP}$ [GFlops]	650	1533	2662	4759	7800	9700
β_{mem} [GB/s]	76.8	119.21	400	732	900	1555



Multi-GPU implementation

- Lattice divided over a set of GPUs
- GPUs virtually arranged in 1D-ring or 2D-grid
- at each step require an additional step ***PBC*** to update halo-columns
- ***PBC*** is a GPU-to-GPU bi-directional memory copy
- 1 MPI-rank per GPU

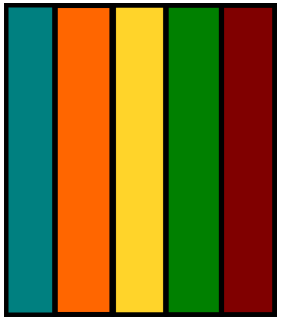


MPI solutions for GPUs:

- MPI standard libraries: OpenMPI, MVAPICH2, ...
- CUDA-aware MPI: integration of CUDA support into MPI (use of GPU-memory pointers)
- MPI derived data-types: specify and handle general data layouts (eg. transfers of non-contiguous data)
- GPUDirect and RDMA: optimization of communications over IB-networks

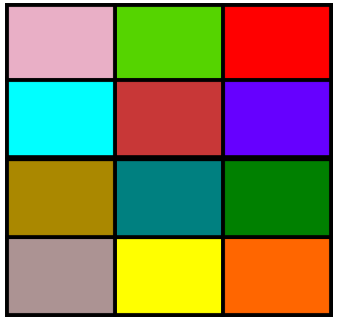
Tiling: 1D vs 2D

1D-Tiling:



- $T_{\text{comm}} \propto 2L$
- $T_{\text{comp}} \propto L * \frac{L}{n} = \frac{L^2}{n}$
- $\frac{T_{\text{comm}}}{T_{\text{comp}}} \propto 2L * \frac{n}{L^2} = \frac{2n}{L}$

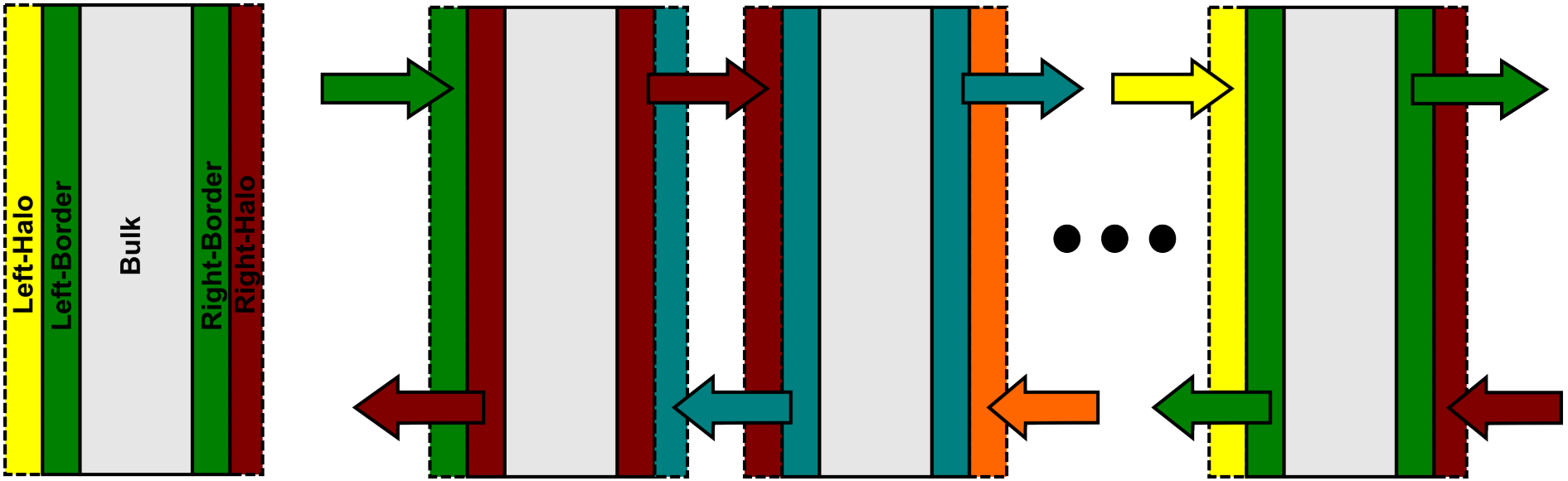
2D-Tiling:



- $T_{\text{comm}} \propto \frac{4L}{\sqrt{n}}$
- $T_{\text{comp}} \propto \frac{L}{\sqrt{n}} * \frac{L}{\sqrt{n}} = \frac{L^2}{\sqrt{n}}$
- $\frac{T_{\text{comm}}}{T_{\text{comp}}} \propto \frac{4L}{\sqrt{n}} * \frac{n}{\sqrt{L^2}} = \frac{4\sqrt{n}}{L}$

$\frac{4\sqrt{n}}{L} < \frac{2n}{L} \quad \Rightarrow \quad$ 2D-tiling should scale better then 1D-tiling.

1D-Tiling:

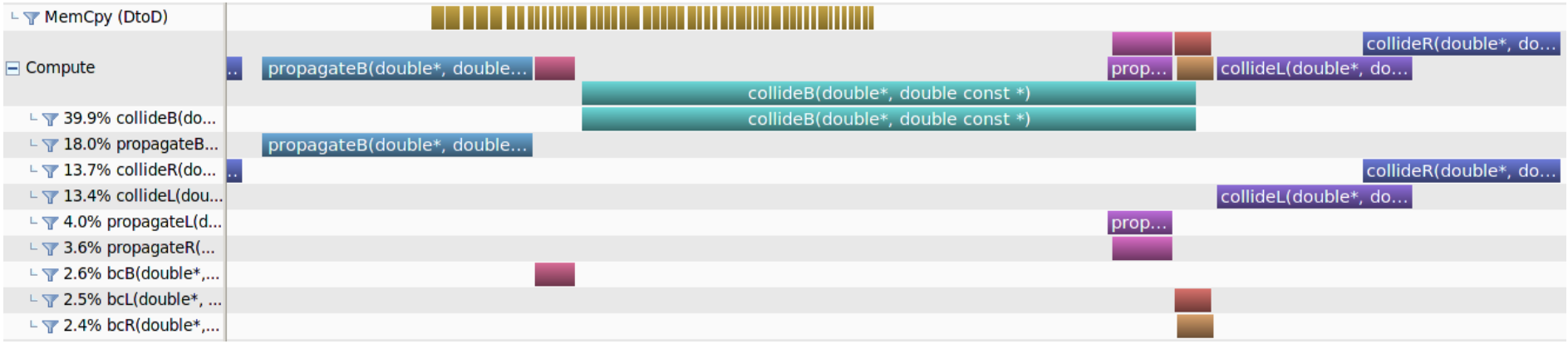


- lattice is divided into sub-lattices each allocated on a different GPU
- each sub-lattice is “divided” in three regions
- GPUs are virtually arrangend on a ring, and each GPU exchanges borders with left- and right-neighbors

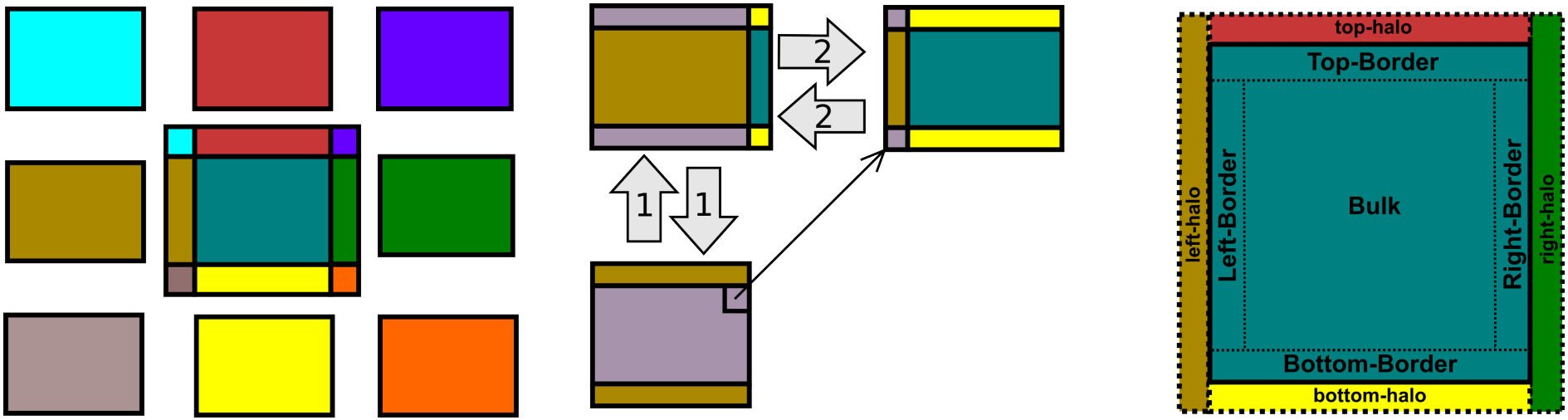
MultiGPU Parallelization via 1D-Tiling

```
// Compute propagate over lattice bulk
prop_Bulk    <<< dimGridB, dimBlockB, 0, stream[0] >>> ( ... );
bc_Bulk      <<< dimGridB, dimBlockB, 0, stream[0] >>> ( ... );
collide_Bulk <<< dimGridB, dimBlockB, 0, stream[0] >>> ( ... );
// Update halo columns
pbc_c();
// Compute propagate on 3 leftmost columns
prop_L       <<< dimGridLR, dimBlockLR, 0, stream[1] >>> ( ... );
bc_L         <<< dimGridLR, dimBlockLR, 0, stream[1] >>> ( ... );
collide_L    <<< dimGridB,  dimBlockB,  0, stream[1] >>> ( ... );
// Compute propagate on 3 rightmost columns
prop_R       <<< dimGridLR, dimBlockLR, 0, stream[2] >>> ( ... );
bc_R         <<< dimGridLR, dimBlockLR, 0, stream[2] >>> ( ... );
collide_R    <<< dimGridLR, dimBlockLR, 0, stream[2] >>> ( ... );
```

Excution timeline on two GPUs attached to single CPU:



2D-Tiling:

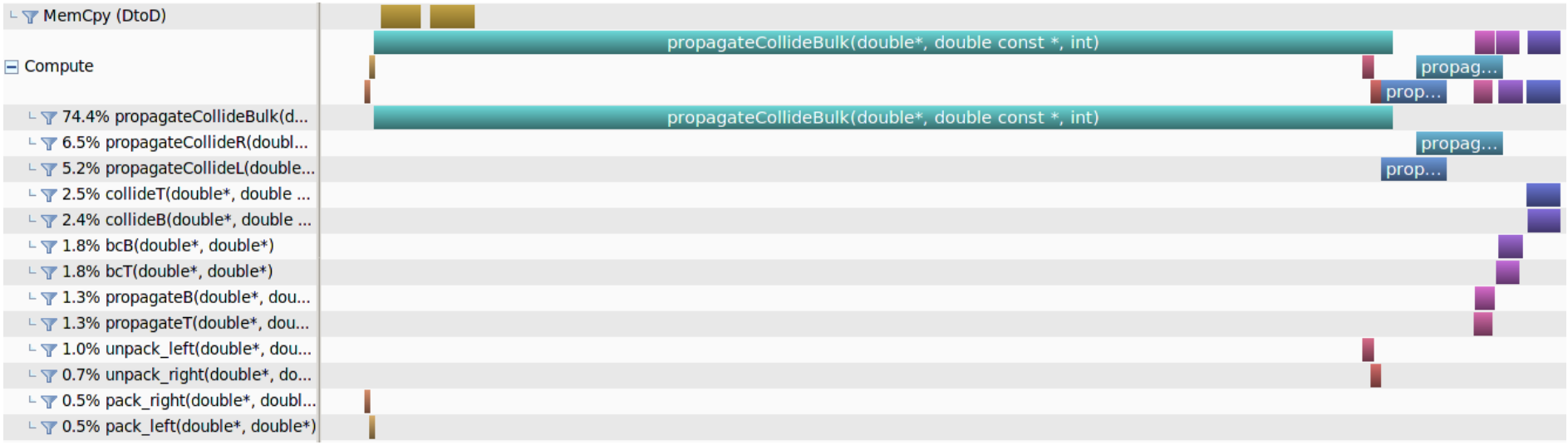


Processing of each sub-lattice is divided in several regions:

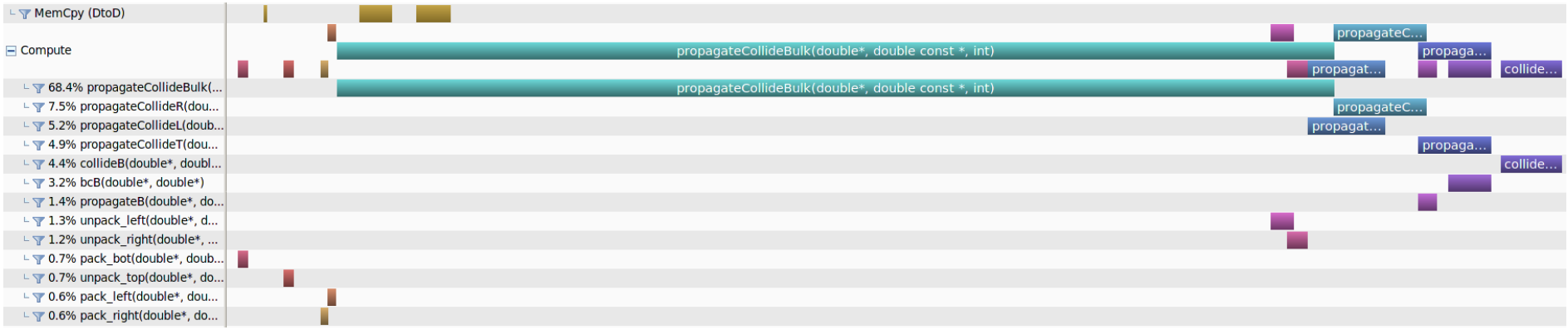
- left- and right-borders (3 columns) are processed after update of left- and right-halos is completed
- top- bottom-borders (3 rows) are processed after udate of top-, bottom-, left- and -right-halos
- bulk does not have dependences with any regions and can be processed in parallel

2D-Tiling: Execution Timeline

Grid 2x1:

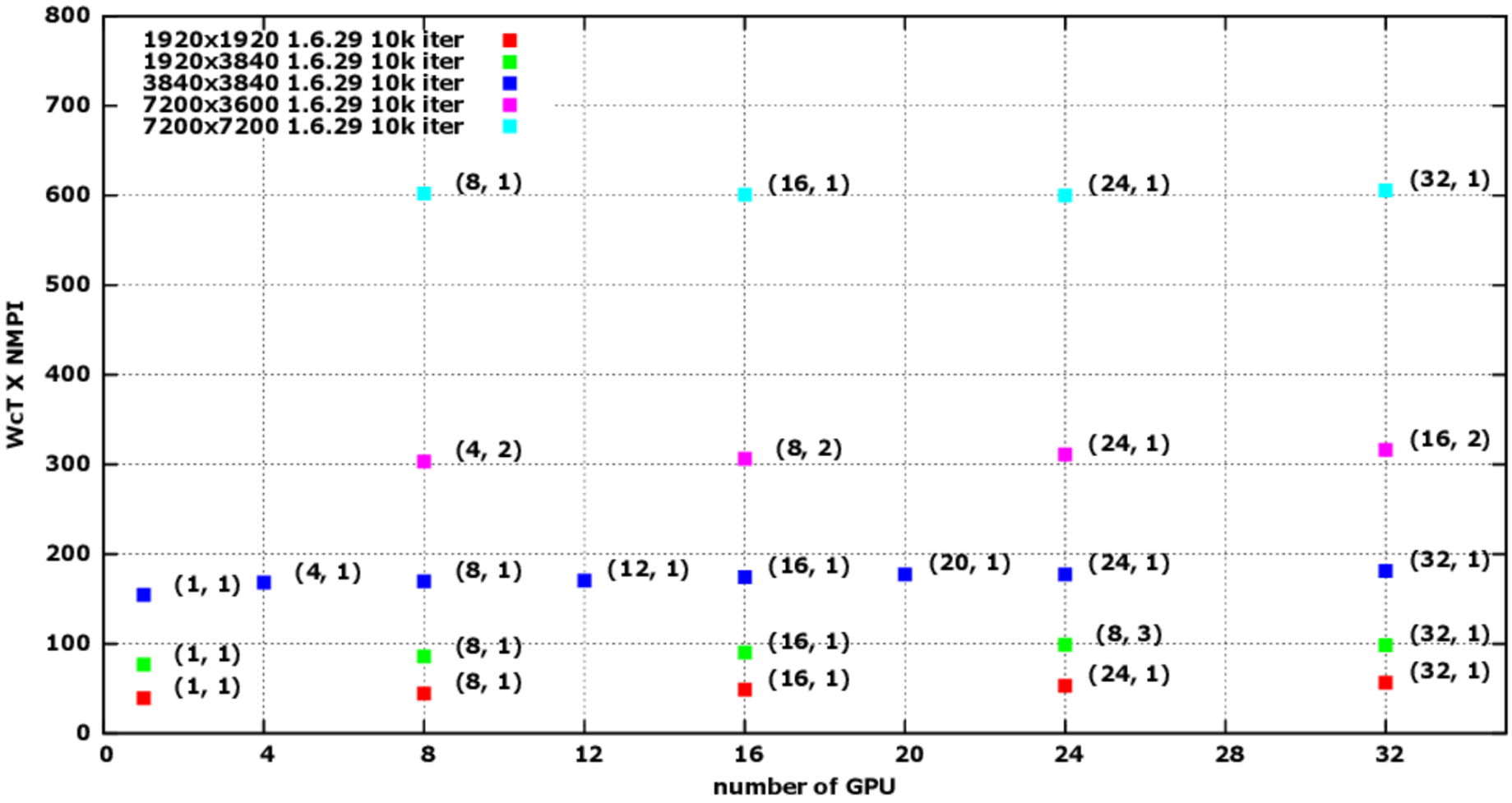


Grid 2x2:



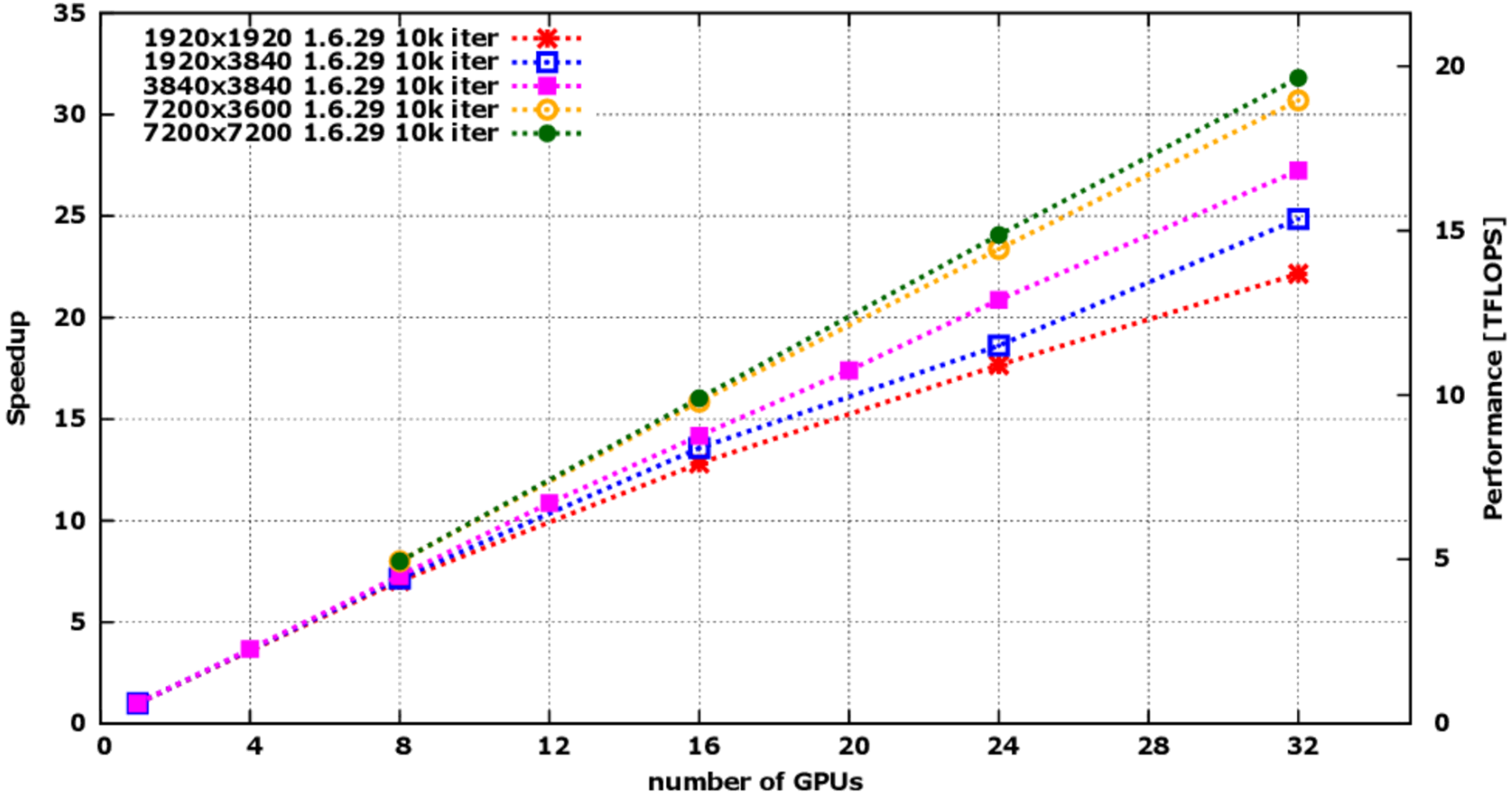
Update of non-contiguous halos can not be overlapped with computation

Performance Scalability



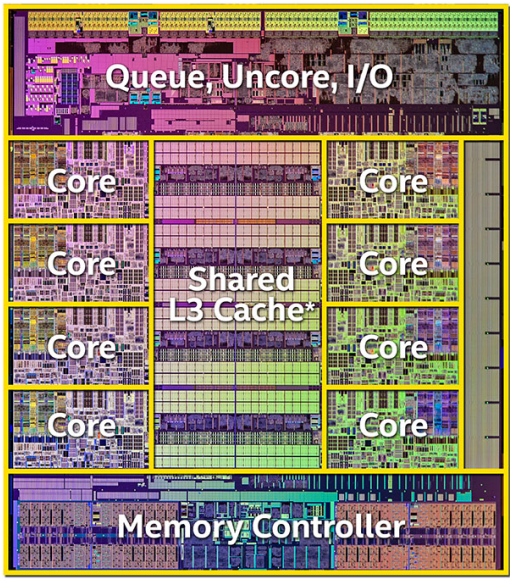
- in most cases the lowest time corresponds to 1D-tiling
- scalability is limited to a small number of GPUs

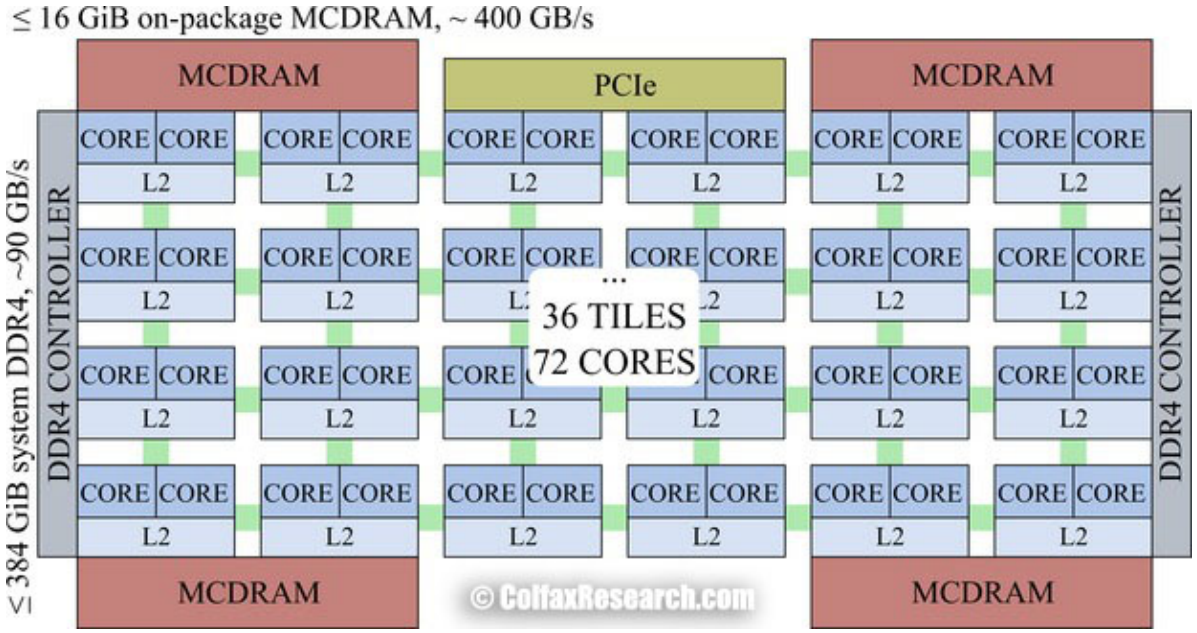
Strong Scaling and Aggregate Performance



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

Multi- and many-core CPUs



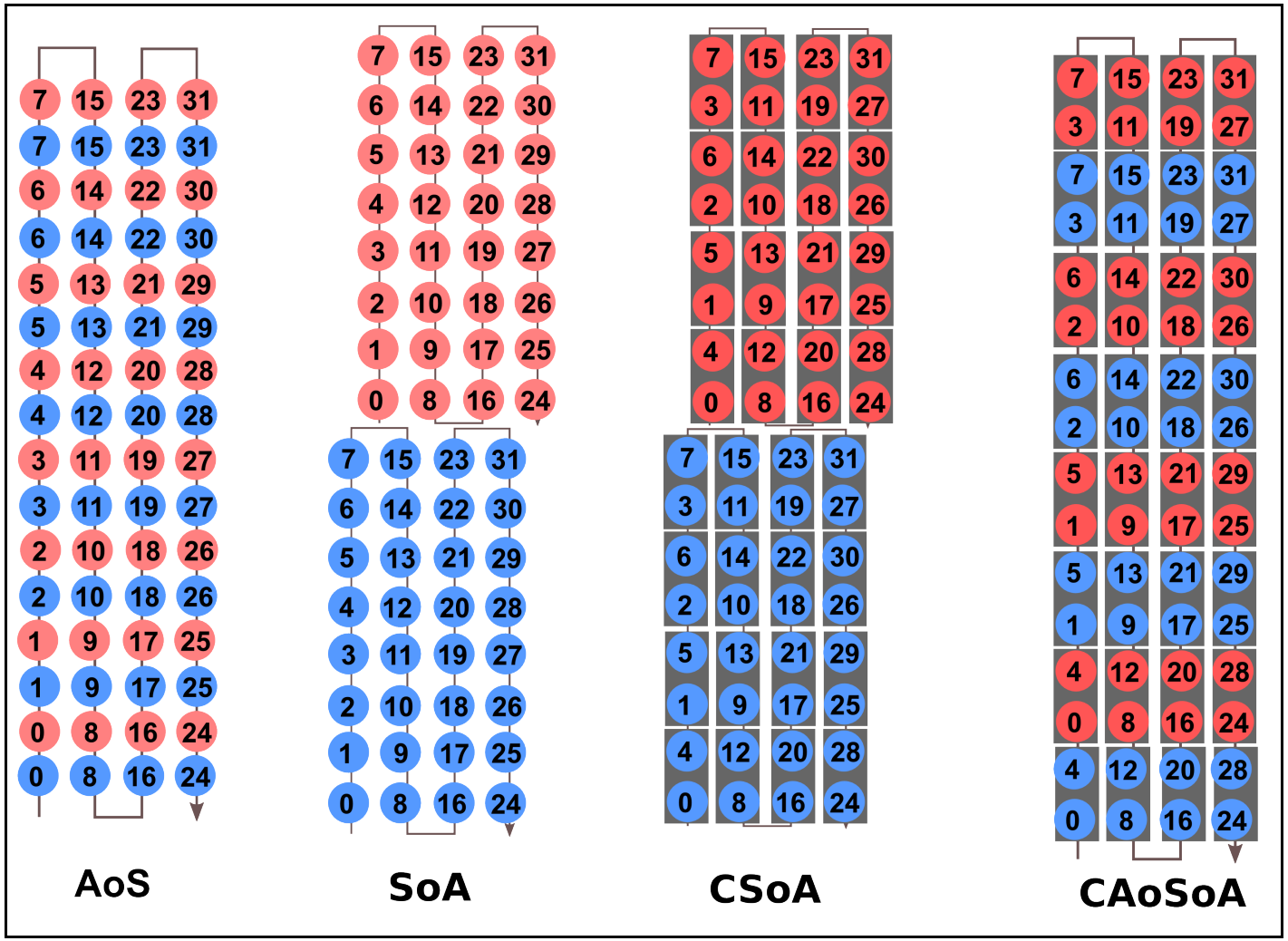


$$P = f \times \#\text{cores} \times \text{nInstrPerCycle} \times \text{nFlopPerInstr}$$

- core parallelism
- hyper-threading
- thread and memory affinity
- vectorization
- cache-awareness
- **data layout**

Memory Layouts for LBM

Lattice 4 $\times$ 8 with two (blu and red) population per site.



Left to right: Array of Structures (AoS), Structure of Arrays (SoA),
Clustered Structure of Arrays (CSoA), Clustered Array of Structure of Arrays (CAoSoA).

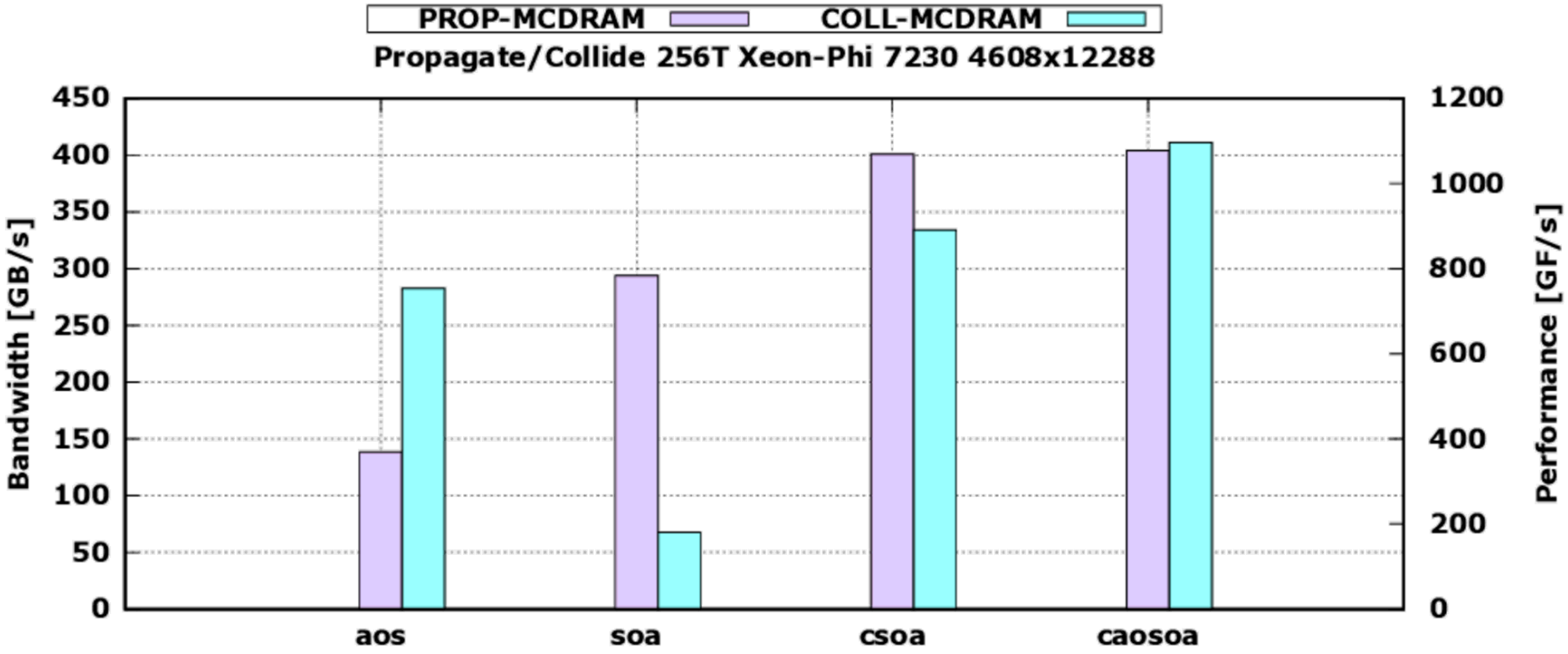
Results: VTUNE Analysis on KNL

Metric	*AoS*	*SoA*	*CSoA*	*CAoSoA*	Threshold
propagate L2 CACHE Miss Rate	0.50	0.10	0.05	0.00	< 0.20
collide 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* (process to recover the page physical-address from page-table)

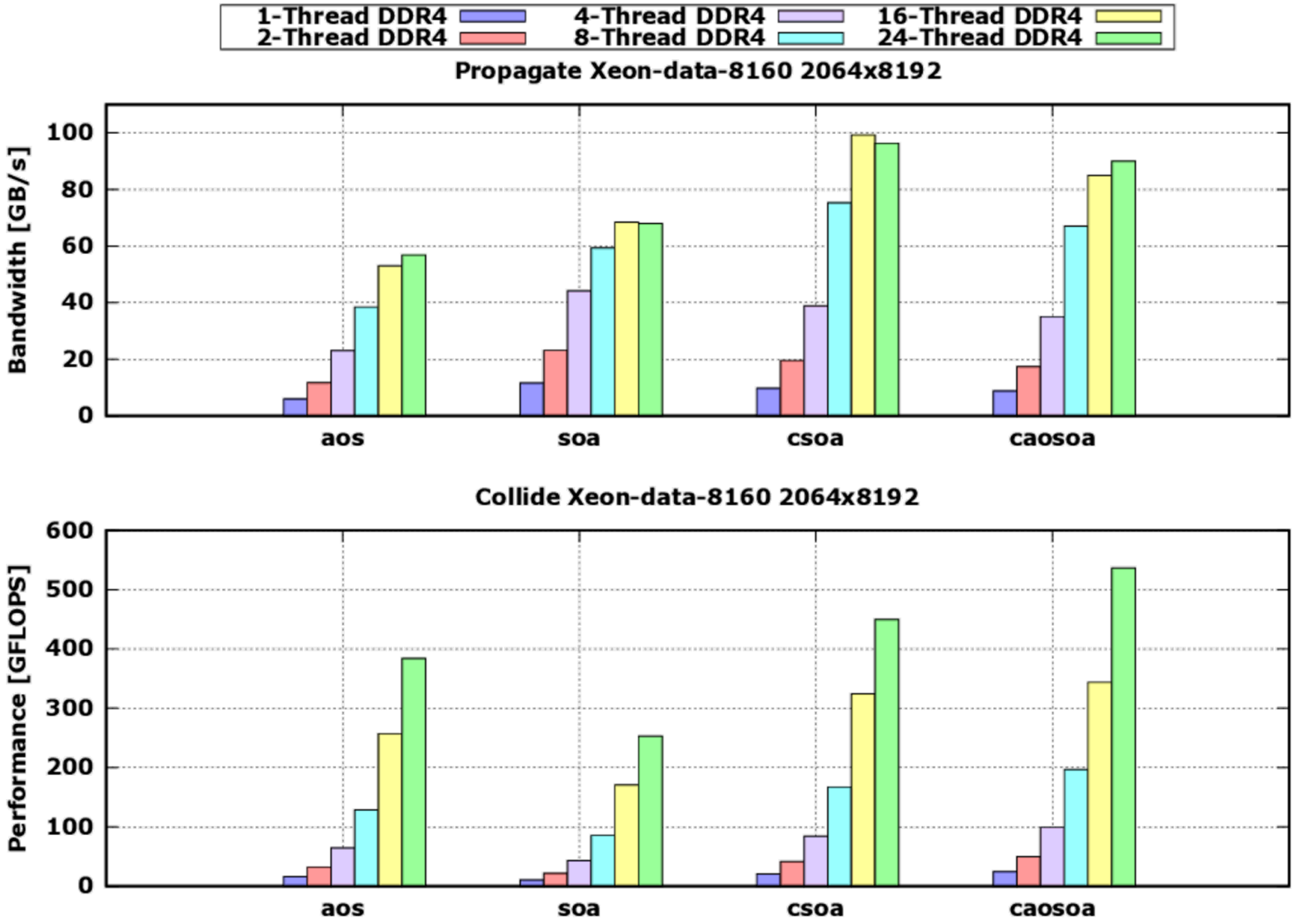
Similar improvements also on Xeon CPU architectures.

Results: Performance on KNL



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

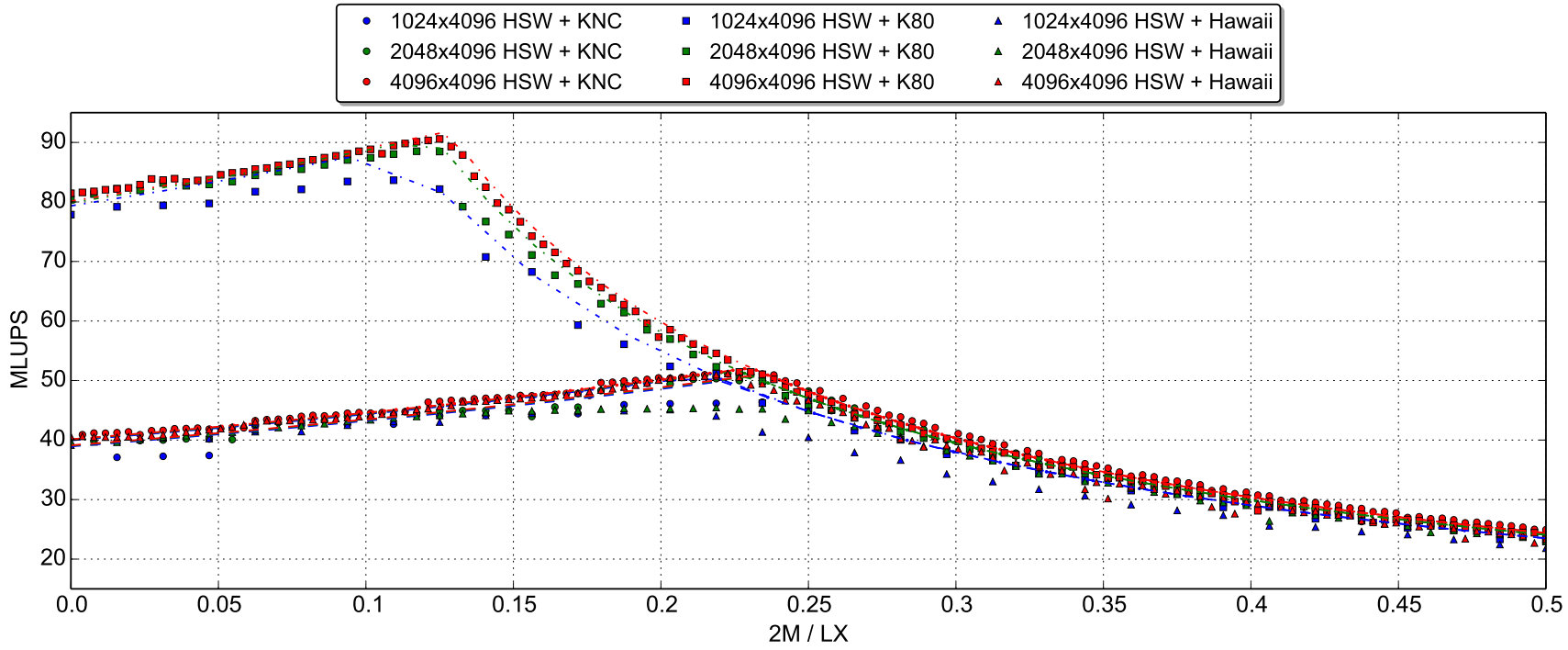
Results: Performance on SkyLake



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

Exploring: Heterogeneous portable computing

One single code both for CPU and GPU.



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

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
- data structure have a big impact on performance
- portability of code and performance is necessary
- GPUs provide great performance
- explore use of latest FPGA accelerators

The D2Q37 application is part of SPEChpc 2021 Benchmark Suite and support CUDA+MPI and OpenACC+MPI.

Acknowledgments

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

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.