Optimization of lattice based simulations for modern HPC architectures

Enrico Calore

University of Ferrara and INFN Ferrara

February 22nd, 2018

BSC, Barcelona, Spain

Table of Contents

Table of Contents

1. Introduction
   - Lattice Boltzmann Methods
   - Lattice Quantum Chromodynamics
2. Lattice Boltzmann D2Q37
   - Jetson Tk1
   - Intel Haswell
   - NVIDIA K80
   - Intel KNL
   - Cavium ThunderX
3. Lattice QCD
   - NVIDIA GPUs and Intel CPUs

Introduction

HPC Scientific Applications

- have to be strongly optimized to exploit the available hardware for high performances;
- but heterogenous processors and architectures are available:

- a solution could be to develop several versions targeting the different architectures

Introduction

But, in scientific applications:

- code undergo to frequent code modifications by scientists ...
  ... hard to maintain different versions;
- is desiderable to have one portable implementation with high performances on most of the available HPC resources;
- scientific software lifetime may be very long; even tens of years, thus is of paramount importance to plan for future architectures.

Introduction

But, in scientific applications:

- code undergo to frequent code modifications by scientists ...
  ... hard to maintain different versions;
- is desiderable to have one portable implementation with high performances on most of the available HPC resources;
- scientific software lifetime may be very long; even tens of years, thus is of paramount importance to plan for future architectures.
- HPC centers may start to account for consumed energy, so energy-efficiency is also becoming an hot-topic

Main motivations

Technology tracking of hardware architectures and programming language/models.

- Study of new architectures' hardware details;
- Study of architecture specific low level optimization techniques;
- Development of higly optimized architecture specific implementations;
- Study of new programming models and languages;
- Development of portable implementations;
- Attempt to foresee future HPC architectures and environments;
- Evaluation of Performance and Energy-efficiency on both specific and portable implementations.

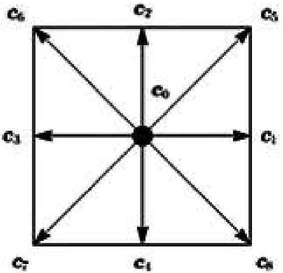
Table of Contents

Lattice Boltzmann Methods (LBM)

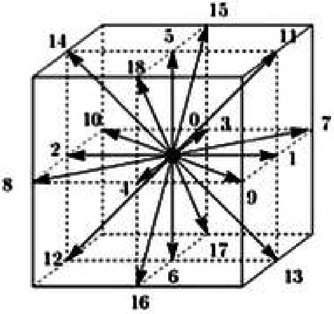
- a class of computational fluid dynamics (CFD) methods
- discrete **Boltzmann** equation instead of **Navier-Stokes** equations
- sets of **virtual particles**, called **populations**, are arranged at edges of a D-dimensional ($D = 2, 3$) lattice
- each population $f_i(x, t)$ has a given fixed lattice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

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

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 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**:

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

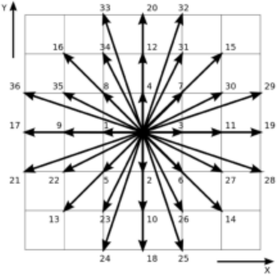
   compute the bulk properties density ρ and velocity $\boldsymbol{u}$, use these to compute the equilibrium distribution $f_i^{(eq)}$, and then relax the fluid distribution functions to the equilibrium state (τ relaxation time).

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

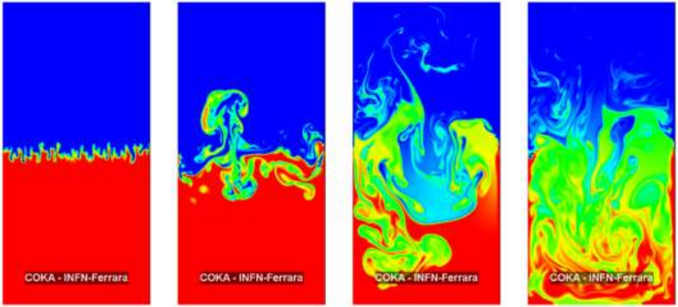


- D2Q37 is a 2D LBM model with 37 velocity components (populations);
- suitable to study behaviour of **compressible** gas and fluids optionally in presence of **combustion**[1] effects;
- include correct treatment of *Navier-Stokes*, heat transport and perfect-gas ($P = \rho T$) equations;
- used to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

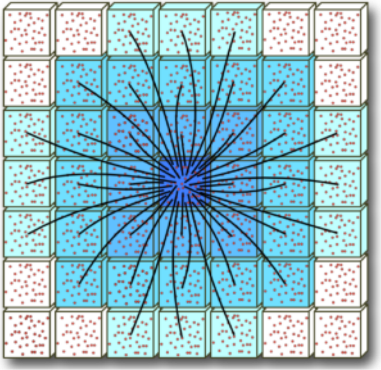


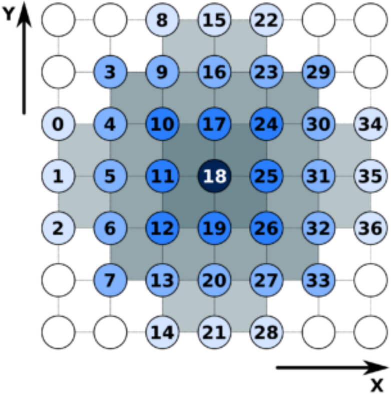
A cold-dense fluid over a less dense and warmer fluid triggers an instability that mixes the two fluid-regions (till equilibrium is reached).

D2Q37: pseudo-code

```
foreach time-step

  foreach lattice-point
    propagate();
  endfor

  boundary_conditions();

  foreach lattice-point
    collide();
  endfor

endfor
```

D2Q37: propagation scheme

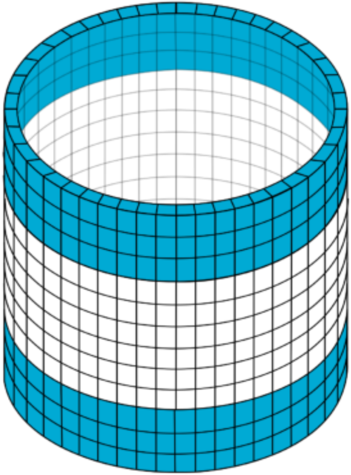




- require to access neighbours cells at distance 1,2, and 3,
- generate memory-accesses with **sparse** addressing patterns.

D2Q37: boundary-conditions

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



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

D2Q37 collision

- collision is computed at each lattice-cell site
- computational intensive:
  for the D2Q37 model requires $\approx$ 6500 DP floating point operations
- computation is completely **local**:
  arithmetic operations require only the populations associated to the site

D2Q37 pseudo-code

```
foreach time—step

  propagate_and_collide_bulk()

  update_halos_LR_halos();

  propagate_and_collide_LR_borders()

  update_halos_TB_halos();

  propagate_top_and_bot();

  boundary_conditions();

  collide_top_and_top();

endfor
```

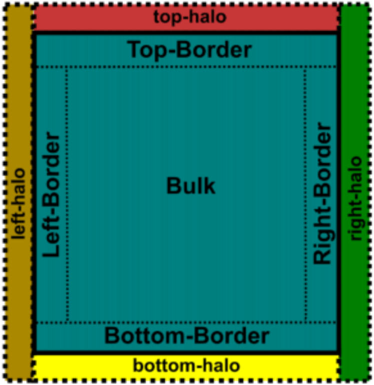


Table of Contents

Lattice Quantum Chromodynamics

Stencil operation on a 4-dimensional Lattice

Most of the running time in a LQCD simulation is spent for the *Dirac Operator*, which executes two functions:

- D_{eo}: reads from even sites of the lattice and writes in odd ones.
- D_{oe}: reads from odd sites of the lattice and writes in even ones.

Both perform mainly complex vector-$SU(3)$ matrix multiplications and are memory-bound operations ($I \approx 1$) with high register pressure.

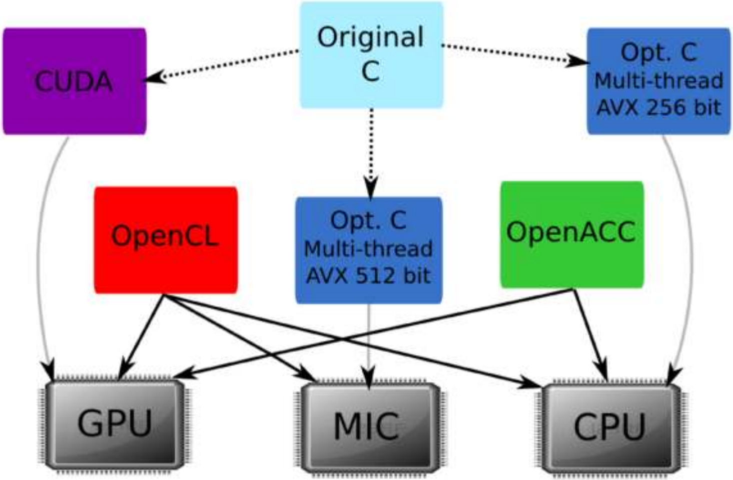
Table of Contents

1 Introduction
- Lattice Boltzmann Methods
- Lattice Quantum Chromodynamics

2 Lattice Boltzmann D2Q37
- Jetson Tk1
- Intel Haswell
- NVIDIA K80
- Intel KNL
- Cavium ThunderX

3 Lattice QCD
- NVIDIA GPUs and Intel CPUs

Different Implementations


Original C
CUDA
Opt. C Multi-thread AVX 256 bit
OpenCL
Opt. C Multi-thread AVX 512 bit
OpenACC
GPU
MIC
CPU

Different Implementations

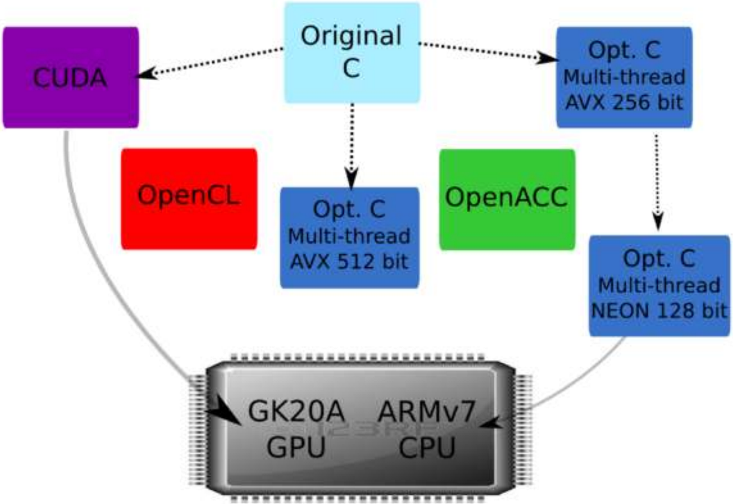
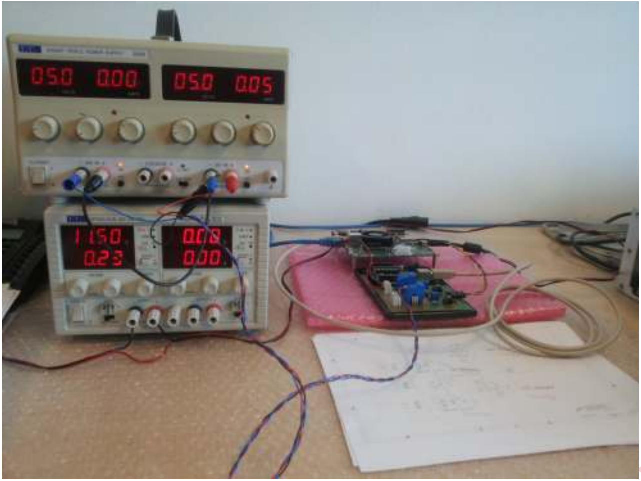

Original C
CUDA
Opt. C Multi-thread AVX 256 bit
OpenCL
Opt. C Multi-thread AVX 512 bit
OpenACC
Opt. C Multi-thread NEON 128 bit
GK20A GPU
ARMv7 CPU

Table of Contents

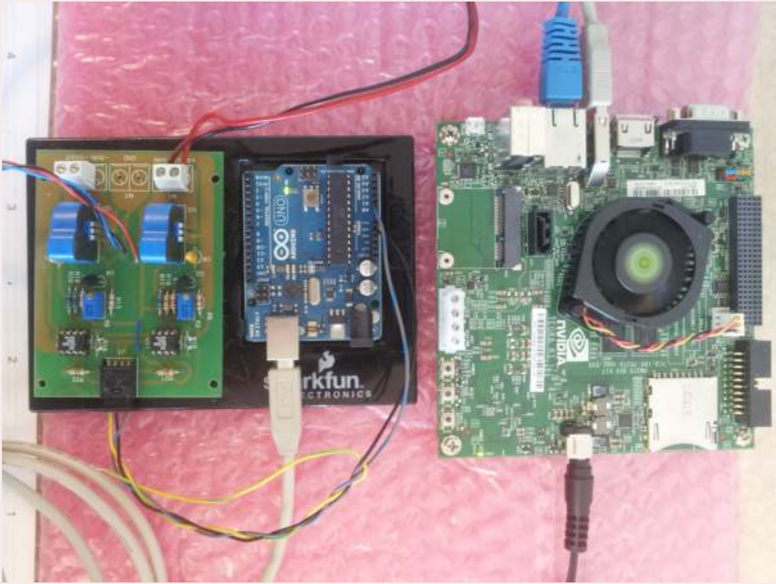
Setup to sample instantaneous current absorption

One current to voltage converter...

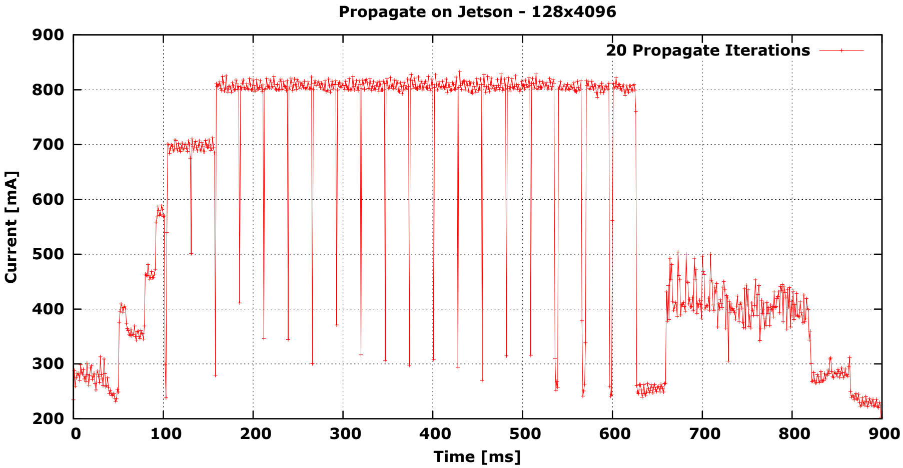
...plus an Arduino UNO (microcontroller + 10-bit ADC + Serial over USB)



Current to Voltage + Digitization with Arduino + USB Serial



Acquired data example with default frequency scaling

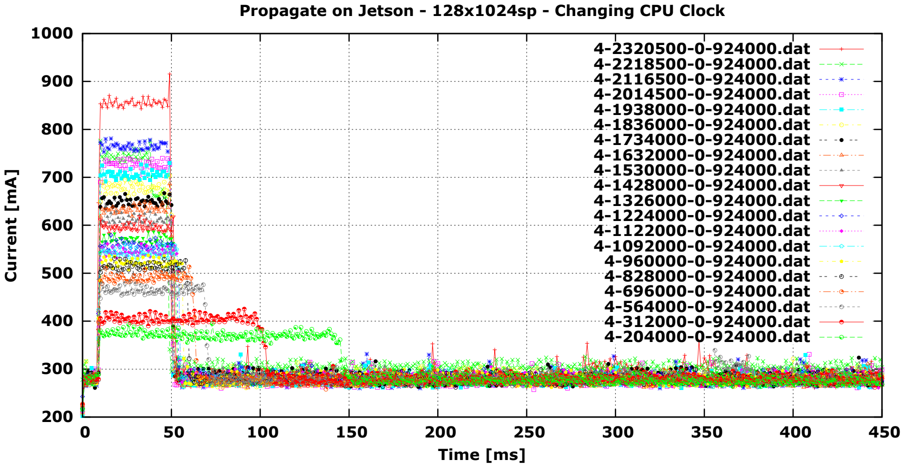


⇑ ⇑ ⇑ . . . ⇑

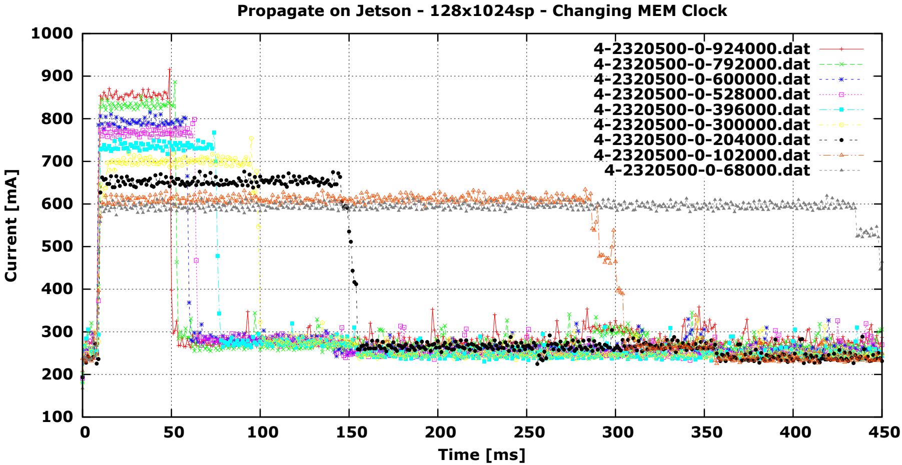
Iterations can be counted

This is a D2H transfer

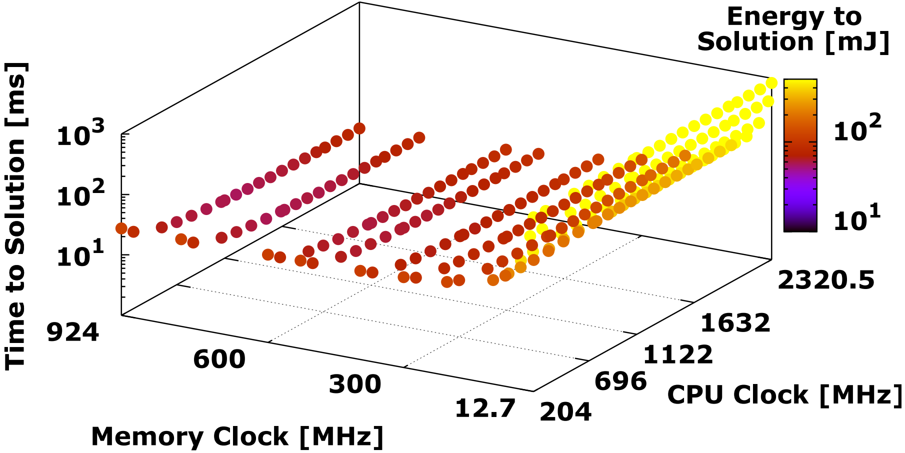
Propagate changing the G cluster clock



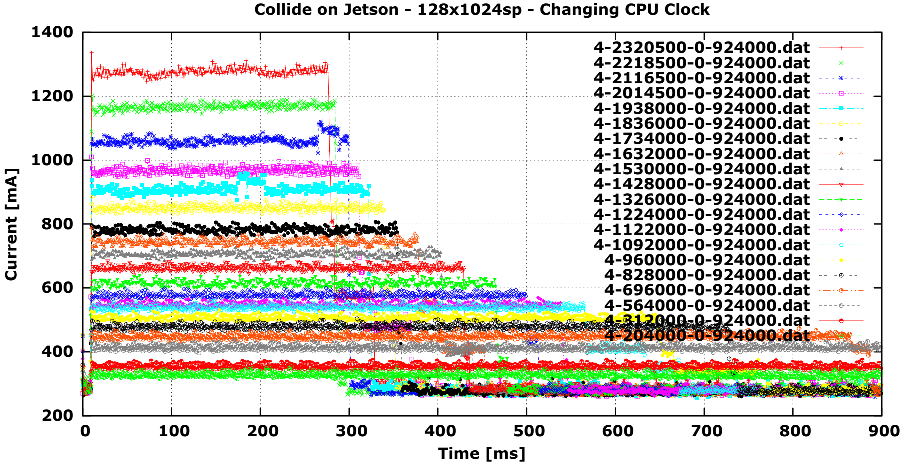
Propagate changing the MEM clock



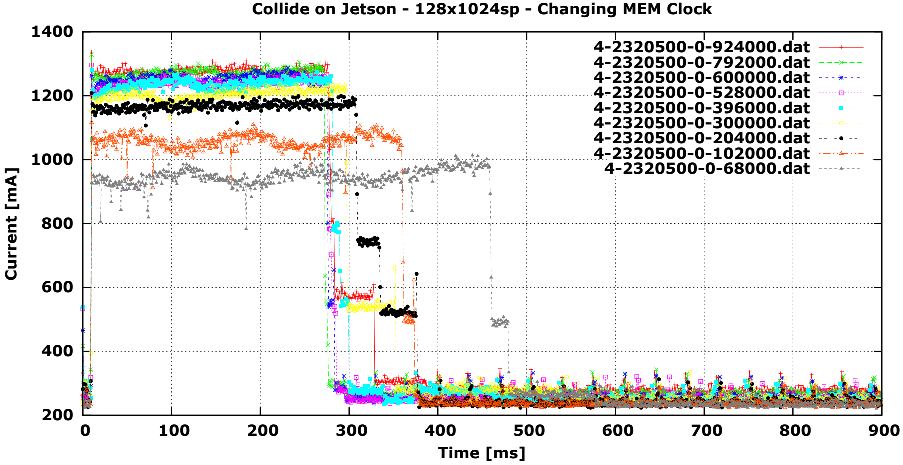
Time and Energy to solution (Propagate)



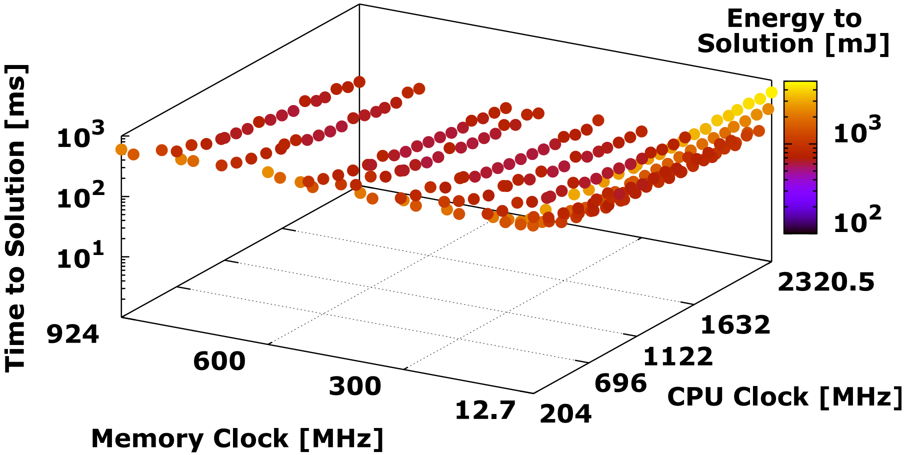
Collide changing the G cluster clock



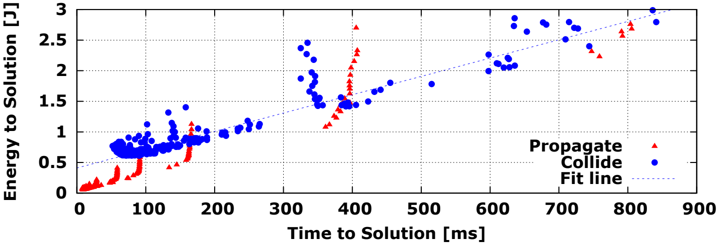
Collide changing the MEM clock

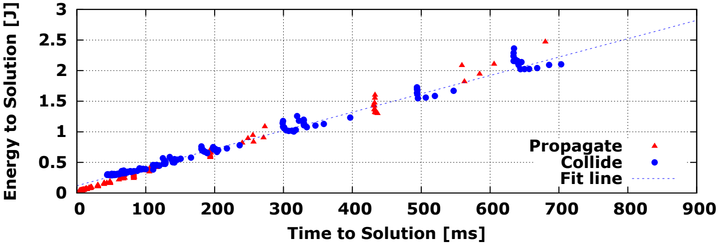


Time and Energy to solution (Collide)

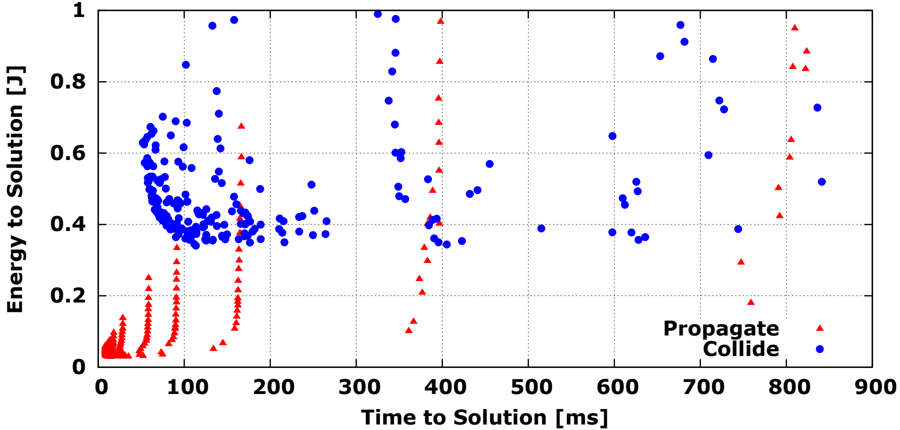


Energy to Sol. vs Time to Sol. CPU(top), GPU(bottom)

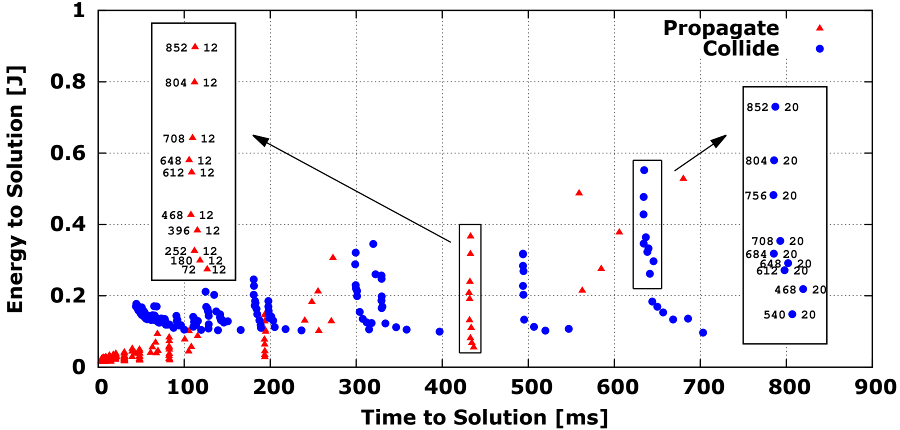




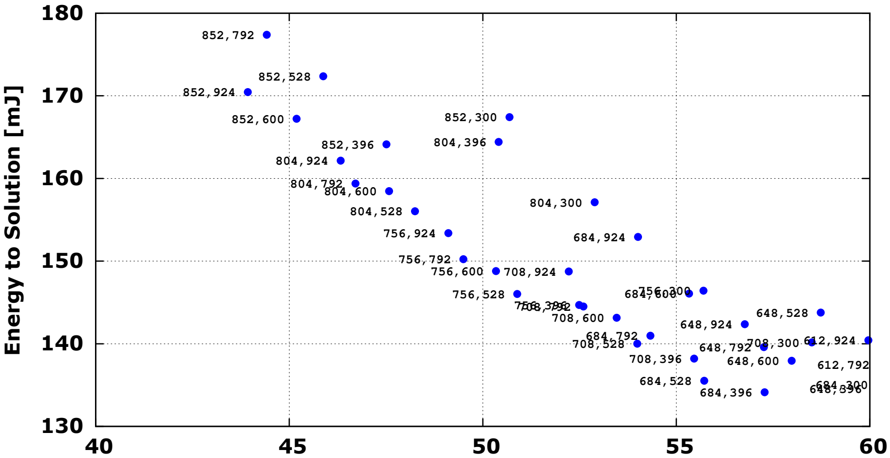
Energy to Solution vs Time to Solution (CPU)



Energy to Solution vs Time to Solution (GPU)



Energy to Solution vs Time to Solution (GPU) zoom

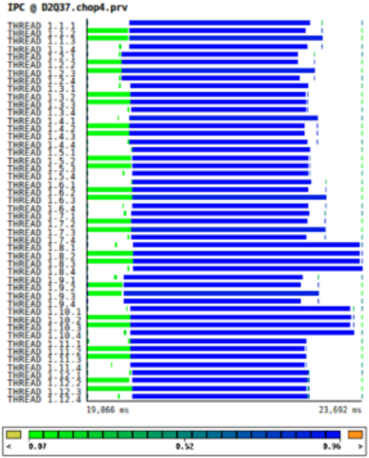


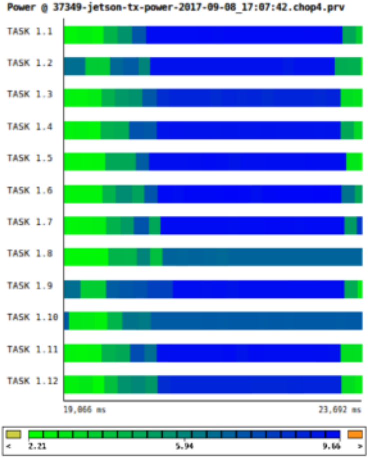
Calore, Enrico and Schifano, Sebastiano Fabio and Tripiccione, Raffaele *Energy-performance tradeoffs for HPC applications on low power processors*, UCHPC15 Workshop at EuroPar, LNCS, 9523, 737-748 (2015). doi: 10.1007/978-3-319-27308-2_59

Conclusions

- baseline power consumption (leakage current + ancillary electronics) is relevant concerning the whole energy budget.
- limited but not negligible power optimization is possible by adjusting clocks on a kernel-by-kernel basis ($\approx$ 20%).
- best region is close to the system highest frequencies.
- options to run the processor at very low frequencies seem almost useless; if possible, it would be interesting to be able to remove power from the (sub-)system while idle.

Ongoing work (Jetson TX Mont-Blanc Cluster)



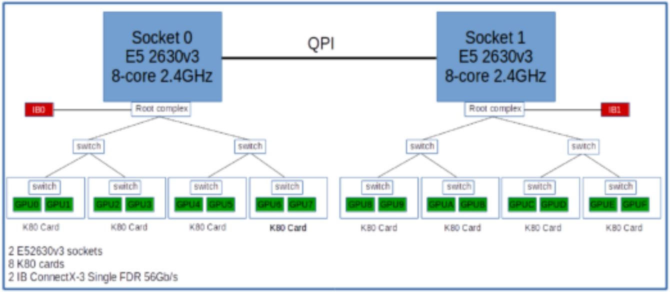


F. Mantovani, E. Calore, *Multi-node advanced performance and power analysis with Paraver*, "Workshop on Energy Aware Scientific Computing on low power and heterogeneous architecture", ParCo2017, (2017). In Press.

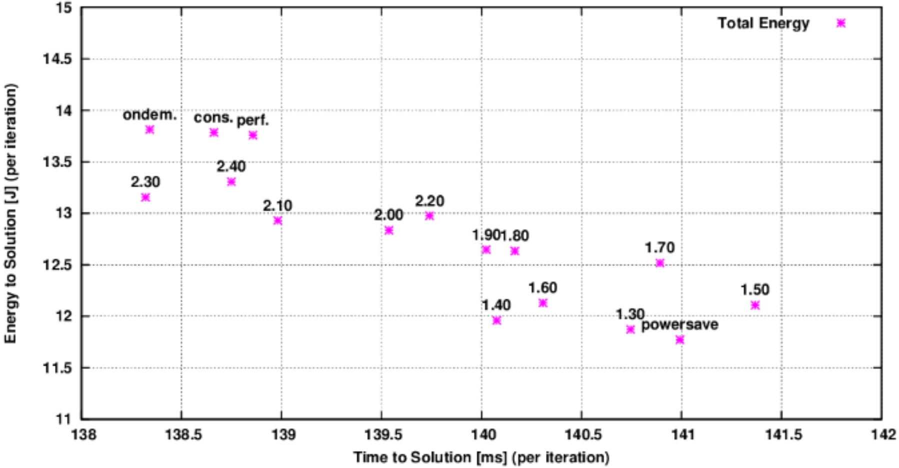
Table of Contents

COKA Cluster

The “Computing On Kepler Architectures” (COKA) is a computing cluster funded and managed by the University of Ferrara with the support of INFN.

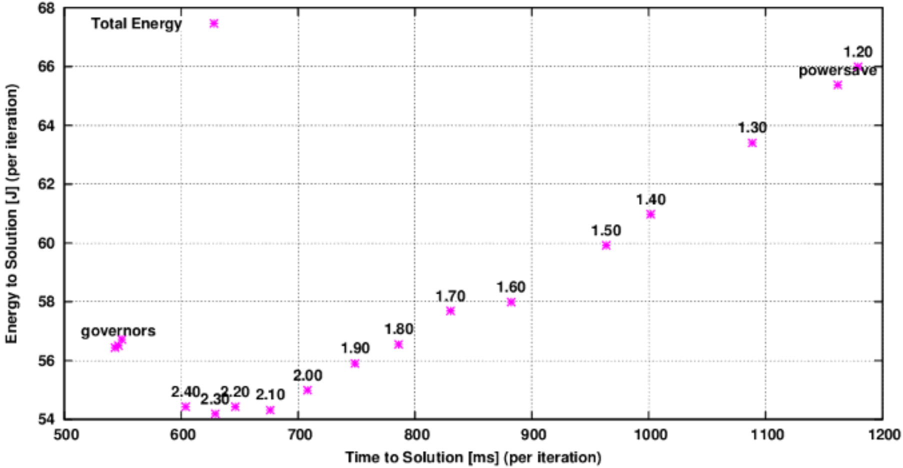


Energy efficiency: Propagate



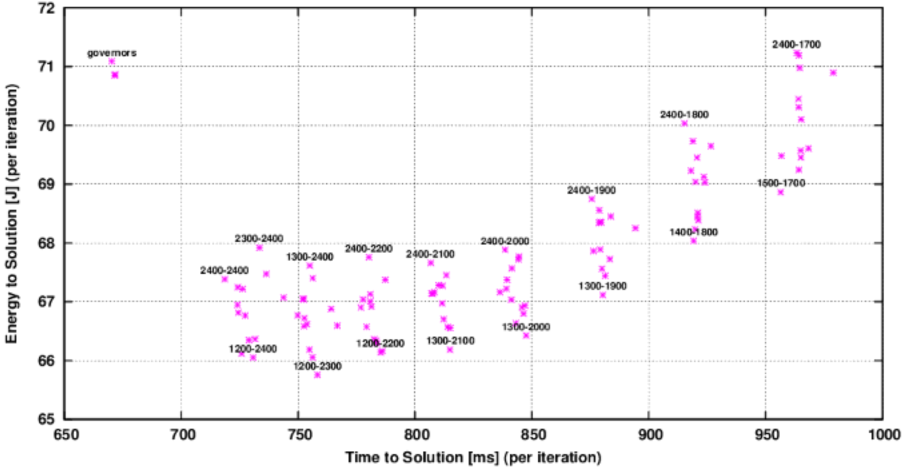
- E_S vs T_S for the propagate functions measured on the CPU;
- labels are the corresponding clock frequencies *f* in GHz.

Energy efficiency: Collide



- E_S vs T_S for the collide functions measured on the CPU;
- labels are the corresponding clock frequencies f in GHz.

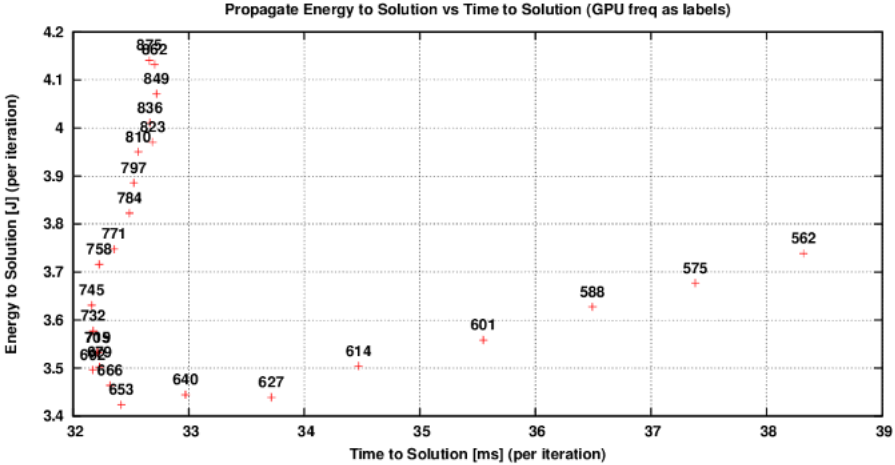
Changing clock function by function



Calore, Enrico and Gabbana, Alessandro and Schifano, Sebastiano Fabio and Tripiccione, Raffaele *Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications*, Concurrency and Computation: Practice and Experience (2017). doi: 10.1002/cpe.4143

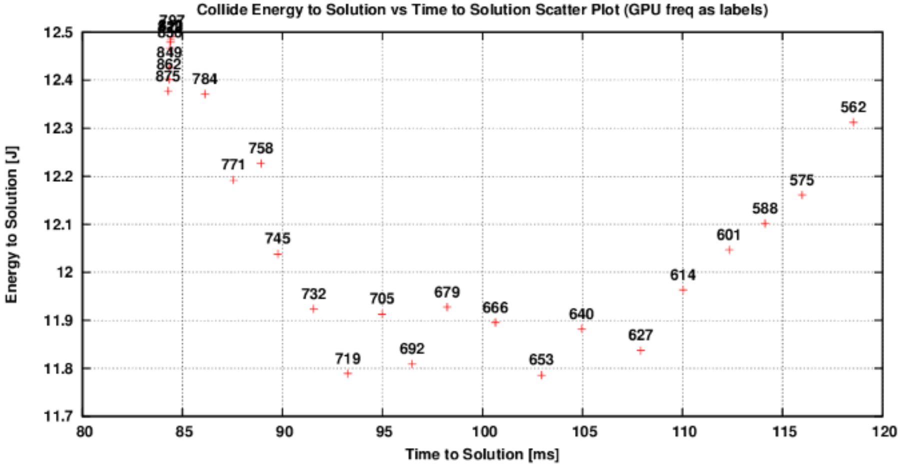
Table of Contents

Energy efficiency: Propagate



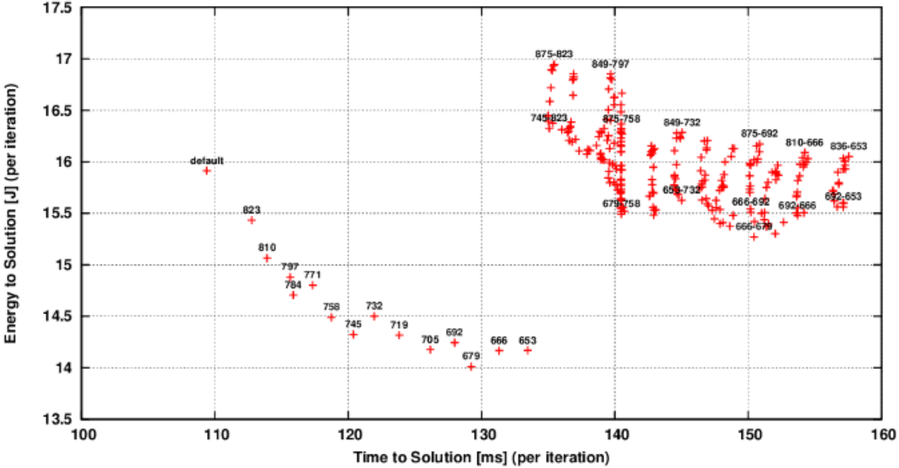
- E_S vs T_S for the propagate functions measured on the GPU;
- labels are the corresponding clock frequencies f in MHz.

Energy efficiency: Collide



- E_S vs T_S for the collide functions measured on the GPU;
- labels are the corresponding clock frequencies f in MHz.

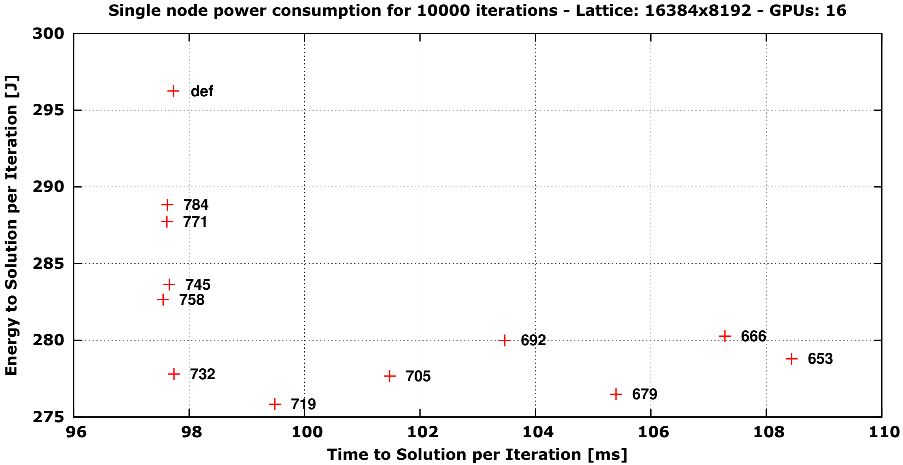
Changing clock function by function



Calore, Enrico and Gabbana, Alessandro and Schifano, Sebastiano Fabio and Tripiccione, Raffaele *Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications*, Concurrency and Computation: Practice and Experience (2017). doi: 10.1002/cpe.4143

Conclusion

Running on 16 GPUs (8 x NVIDIA K80 Dual GPU boards) system:



Power drain of the node measured at PSU through IPMI, during code execution for different GPUs clock frequencies.

Table of Contents

Xeon-Phi KNL

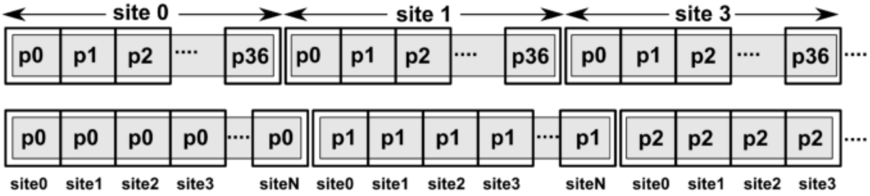
KNL is the latest generation of processors based on Intel MIC architecture:

- 64-68-72 CPU cores
- 512-bit vector instructions
- 3+ Tflops DP peak floating-point
- 16 GB on-chip memory with $400+$ GB/s of bandwidth
- up to 384 GB of off-chip DDR4 memory with $\approx$ 115 GB/s of bandwidth

	Xeon E52697v4	GK210	P100	Xeon-Phi 7120P	Xeon-Phi 7230
Year	2016	2014	2016	2013	2016
Architecture	Broadwell	Kepler	Pascal	Knights Corner	Knights Landing
#physical-cores / SMs	18	13	56	61	64
#logical-cores / CUDA-cores	26	2496	3584	244	256
Nominal Clock (GHz)	2.3	562	1.3	1.2	1.3
Nominal DP performance (Gflops)	$\approx$ 650	935	4759	$\approx$ 1208	$\approx$ 2662
LL cache (MB)	45	1.68	4	30.5	16000
Total memory supported (GB)	1540	12	16	8	384
Peak mem. BW (ECC-off) (GB/s)	76.8	240	732	352	115.2

GK210 is one GPU-processor of a dual-GPU K80 board.

Memory Layout: AoS vs SoA


site 0 | p0 | p1 | p2 | .... | p36 | site 1 | p0 | p1 | p2 | .... | p36 | site 3 | p0 | p1 | p2 | .... | p36 | ....

p0 site0 | p0 site1 | p0 site2 | p0 site3 | .... | p0 siteN | p1 site0 | p1 site1 | p1 site2 | p1 site3 | .... | p1 siteN | p2 site0 | p2 site1 | p2 site2 | p2 site3 | ....

```
#define N (LX*LY)
typedef struct {
  double p1; // population  1
  double p2; // population  2
  ...
  double p37; // population 37
} pop_t;

pop_t lattice[N];
```

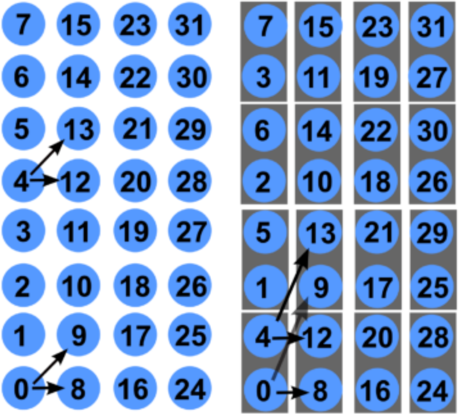
```
#define N (LX*LY)
typedef struct {
  double p1[N]; // population  1
  double p2[N]; // population  2
  ...
  double p37[N]; // population 37
} pop_t;

pop_t lattice;
```

- data arrangement layouts: AoS (upper), SoA (lower);
- `C-struct` data types: AoS (left), SoA (right).

Data Structure: SoA vs CSoA

Straight SoA-scheme does not vectorize propagate properly generating many not-aligned loads.

- Lattice 4 × 8
- machine vector size of 2-doubles
- 8 Bytes memory alignement
- process two sites in parallel
- 0 → 8 has read and write aligned
- 0 → 9 has read aligned and write mis-aligned
- (0, 4) → (8, 12) has read and write aligned
- (0, 4) → (9, 13) has read and write aligned
- clusters close to borders need special handling

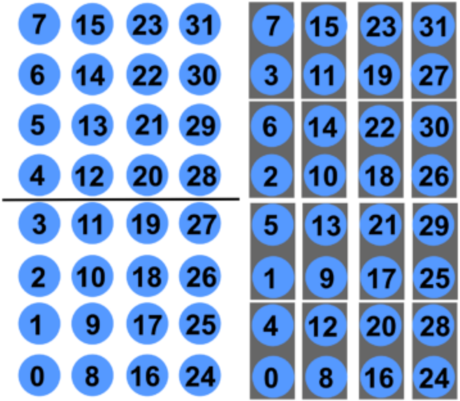


Solution: CSoA layout

Rearrange populations to apply propagate on clusters/vectors instead of a single lattice-cell.

Data Structure CSoA

- for a lattice of size $LX \times LY$
- cluster together `VL` elements of each population at a distance LY / VL
- `VL` is a multiple of the processor vector size.



Using *CSoA* layout *propagate* is fully vectorized with aligned memory accesses.

Data Structure: CAoSoA

- Using CSoA data-layout code for *collide* is properly vectorized but performance are low;

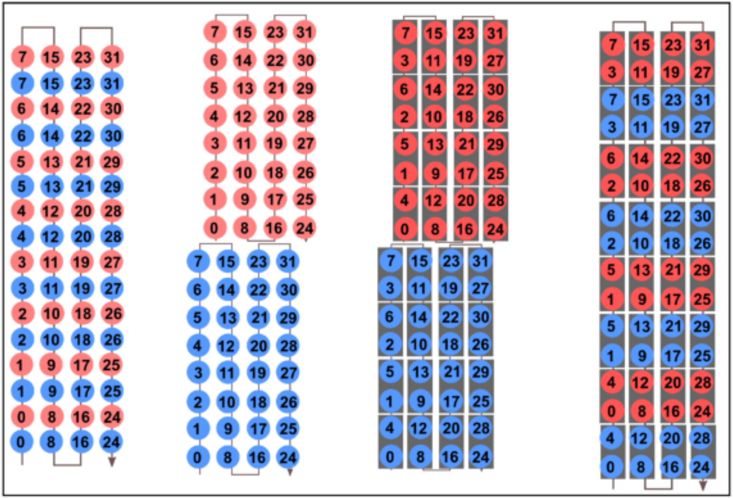
- many TLB misses in executing the *collide* kernel are caused by several strided memory accesses to load all data populations to compute the collisional operator.

Solution: CAoSoA layout

- for each population array, we divide each Y-column in `VL` partitions each of size `LY/VL`
- all elements sitting at the ith position of each partition are then packed together into an array of `VL` elements called *cluster*.
- For each index i we then store in memory one after the other the 37 clusters – one for each population – associated to it keeping all population data associated to each lattice site at close and aligned addresses.

Data Structure

Lattice 4 × 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).

Motivations behind memory layouts

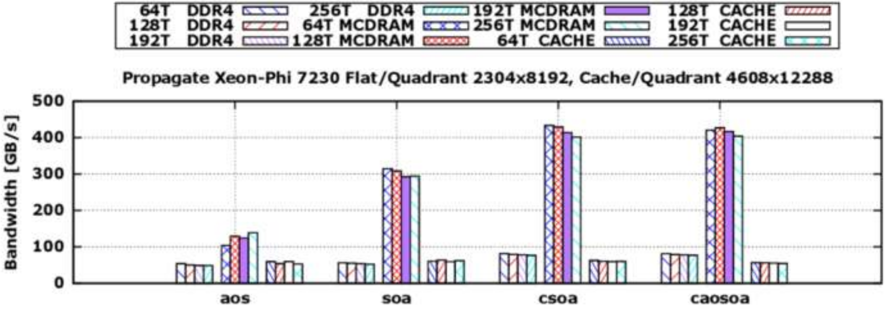
- *AoS*: Array of Structures
  - store populations of one site close in memory
  - store sites one after the other
  - exploit locality of populations, suitable for *collide* does not allow to vectorize *propagate*.
- *SoA*: Structure of Arrays
  - for each population index store all sites one after the other
  - store different populations on different arrays
  - not allow optimal vectorization of *propagate* because of misaligned memory accesses.
- *CSoA*: Clustered Structure of Arrays
  - same as AoS
  - vectorize *propagate* but have negative impact on *collide* since populations are far from each other.
- *CAoSoA*: Clustered Array of Structures of Arrays
  - mix layout
  - inner structure stores populations in SoA format
  - external structure store sites in Aos format
  - vectorize *propagate* and *collide* and improve populations-locality.

Results: VTUNE Analysis

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

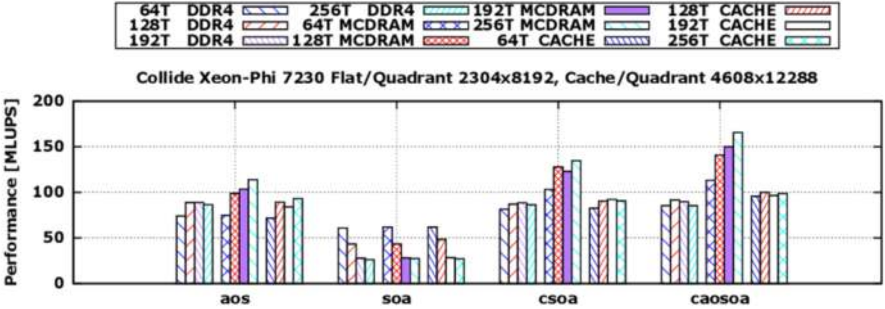
Thresholds suggested by the Intel VTUNE profiler.

Results: Propagate Performance



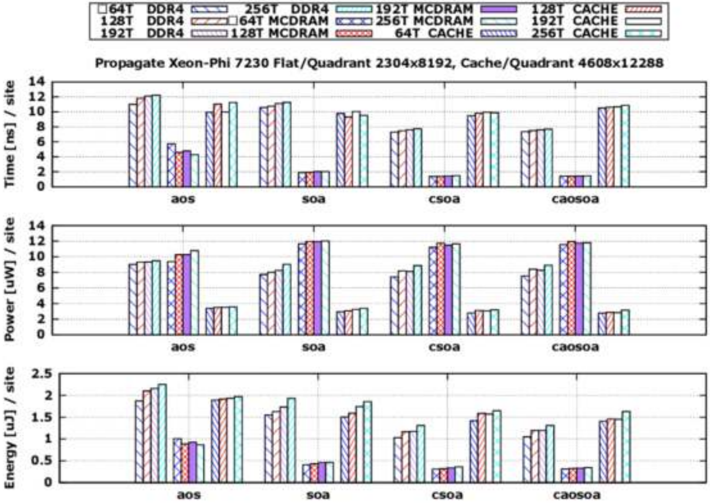
- for FLAT-mode performance increases from *AoS* → *SoA* → *CSoA* with peak bandwidths of
  - MCDRAM: *AoS* 138, *SoA* 314 (2.3X), *CSoA* 433 (3.1X) GB/s
  - DDRA4: *AoS* 51, *SoA* 56, and *CSoA* 81 GB/s
- for CACHE-mode we measure 59, 60 and 62 GB/s with a lattice not fitting into MCDRAM
- performance does not improve
  - using the *CAoSoA* layout
  - increasing number of threads used (since *propagate* is memory-bound)

Results: Collide Performance



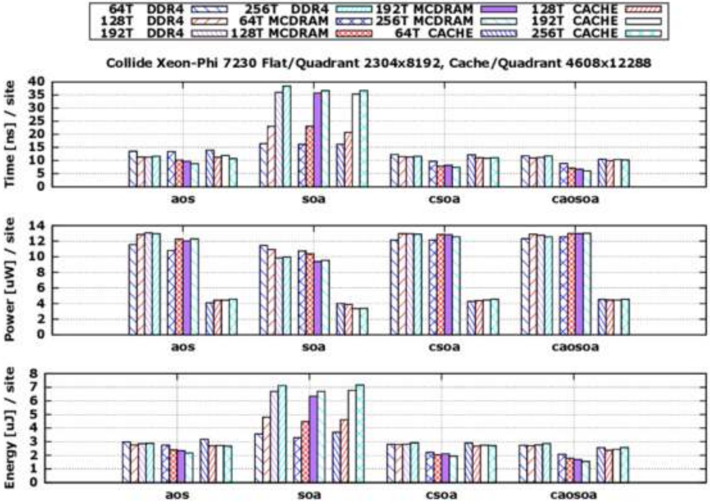
- for FLAT-MCDRAM configuration
  - performance increases from *AoS* → *CSoA* → *CAoSoA*
  - *SoA* does not exploit vectorization and memory-alignement
  - using *CAoSoA* we measure a performance of ≈ 1 Tflops (≈ 37% of raw peak)
  - performance increases with number of threads because *collide* is compute-bound
- using FLAT-DDR4 and CACHE configurations performances are limited by memory bandwidth

Results: Propagate Energy



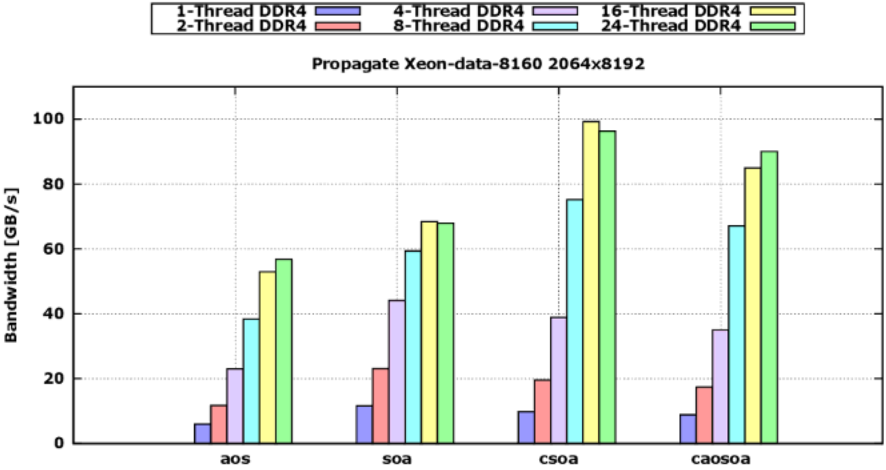
- power-drain as sum of processor and DDR4 measured using RAPL counters
- *CSoA* gives the best E_S, $\approx$ 2.5X lower w.r.t. *AoS* for FLAT-MCDRAM configuration

Results: Collide Energy



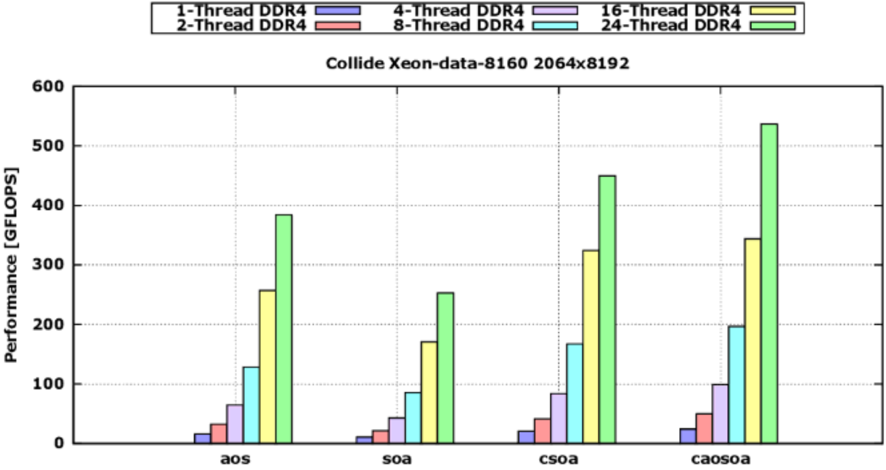
- *CAoSoA* gives the best E_S, ≈ 2X lower w.r.t. *AoS* for FLAT-MCDRAM configuration
- E_S decreases using more threads per CPU

Results: Propagate Performance on SkyLake



propagate $\approx$ 100 GB/s, *approx*85% of raw peak.

Results: Collide Performance on Skylake



collide ≈ 530 GFlops. ≈ 35% of raw peak.

Conclusions

- KNL architecture makes it easy to port and run codes previously developed for X86 standard CPUs;
- performance is strongly affected by massive level of parallelism that must be exploited, lest the level of performance drops to values of standard multi-core CPUs or even worst;
- data layouts plays a relevant role to enable energy-efficiency and performance;
- if application data-domain fits within the MCDRAM, energy-efficiency and performance are very competitive with GPU accelerators;

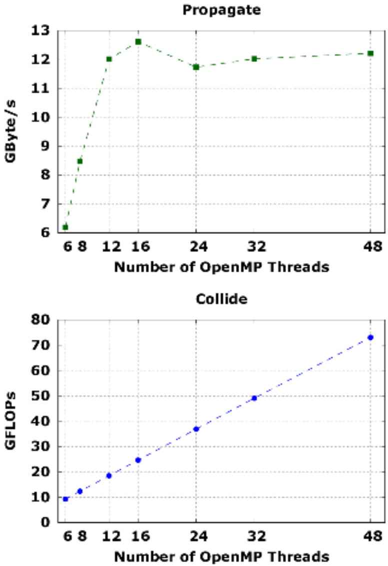
E. Calore, A. Gabbana, S. F. Schifano, R. Tripiccione *Early experience on using Knights Landing processors for Lattice Boltzmann applications*, “Parallel Processing and Applied Mathematics: 12th International Conference”, PPAM (2017). In Press.

E. Calore, A. Gabbana, S. F. Schifano, R. Tripiccione *Energy-efficiency evaluation of Intel KNL for HPC workloads*, “Workshop on Energy Aware Scientific Computing on low power and heterogeneous architecture”, ParCo2017, (2017). In Press.

Table of Contents

2. Lattice Boltzmann D2Q37
   - Cavium ThunderX

OpenMP version with ARMv8 NEON Intrinsics

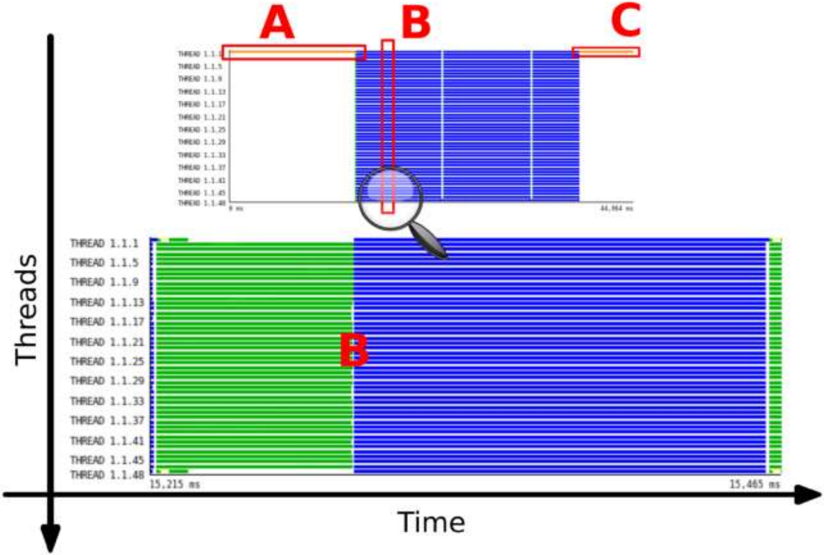


Thunder cluster at BSC, Mont-Blanc 2 project.

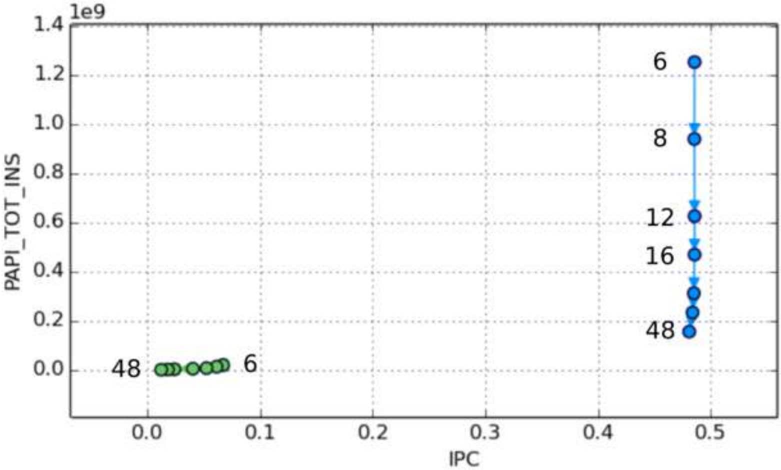
Cavium ThunderX Pass2 SoC.

- STREAM benchmark reach: 39.6 GB/s
- Theoretical peak performance is 192 GFLOPs

Profiling with Extrae and Paraver

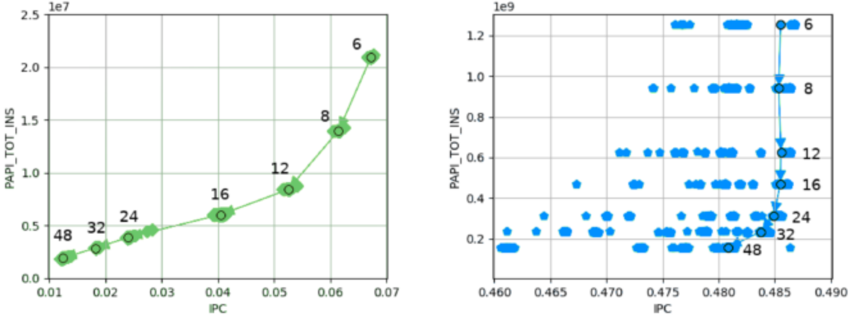


Clustering and Tracking



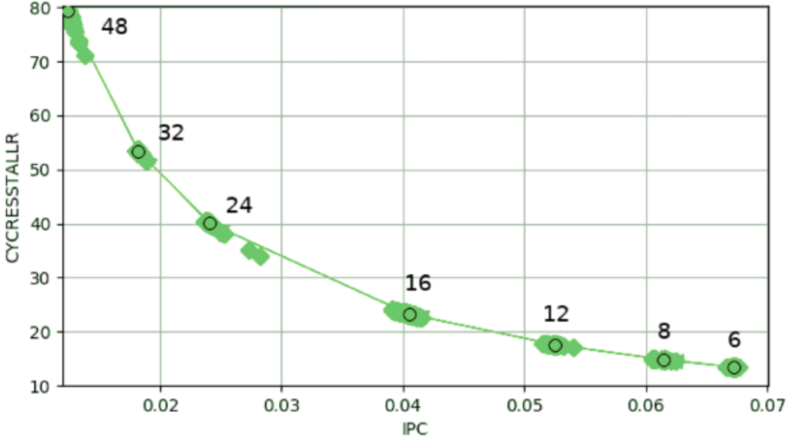
propagate on the left and *collide* on the right

Clustering and Tracking



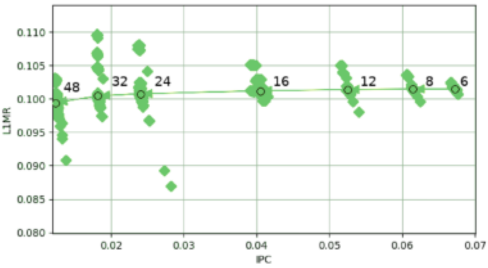
Zoom over the two functions,
propagate on the left and *collide* on the right.

Clustering and Tracking (Propagate)

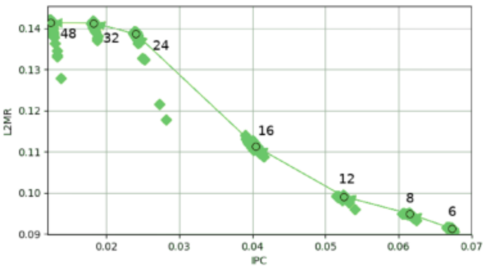


Clusters movement in the Cycle-wasted-in-resource-stall-ratio vs IPC

Clustering and Tracking (Propagate)

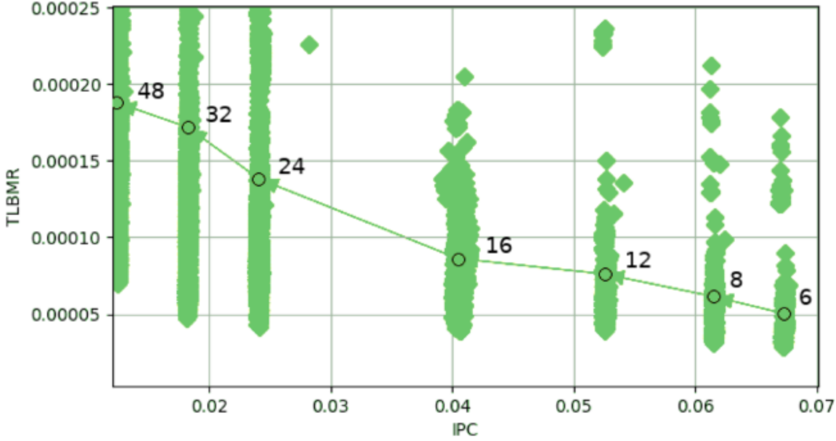


L1 Cache Miss Ratio



L2 Cache Miss Ratio

Clustering and Tracking (Propagate)



Clusters movement in the TLB miss Ratio vs IPC

Conclusion

- As for the KNL, the data structure holding the Lattice can be optimized to reduce L2 and TLB cache misses.
- Preliminary results adopting the *CAoSoA* data layout show a factor 2 improvement in the *propagate* bandwidth.

Table of Contents

1. Introduction
   - Lattice Boltzmann Methods
   - Lattice Quantum Chromodynamics
2. Lattice Boltzmann D2Q37
   - Jetson Tk1
   - Intel Haswell
   - NVIDIA K80
   - Intel KNL
   - Cavium ThunderX
3. Lattice QCD
   - NVIDIA GPUs and Intel CPUs

Using OpenACC towards code portability

The case of Lattice QCD

- Existing versions of the code targeting different architectures:

  ⇒ C++ targeting x86 CPUs

  ⇒ C++/CUDA targeting NVIDIA GPUs

The faced challenge is to design and implement one version:

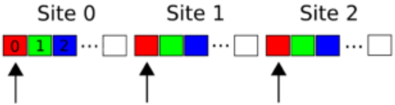
- with good performances on present best performing architectures;
- portable across different available architectures;
- easy to maintain, allowing scientists to change/improve the code;
- portable, or easily portable on future unknown architectures.

Table of Contents

Planning the memory layout for LQCD : AoS vs SoA

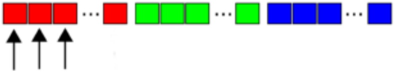
First version in C++ targeting CPU based clusters adopts **AoS**:

```
//fermions stored as AoS:
typedef struct {
  double complex c1; // component 1
  double complex c2; // component 2
  double complex c3; // component 3
} vec3_aos_t;

vec3_aos_t fermions[sizeh];
```



Version in C++/CUDA targeting NVIDIA GPU clusters adopts **SoA**:

```
//fermions stored as SoA:
typedef struct {
  double complex c0[sizeh]; // components 1
  double complex c1[sizeh]; // components 2
  double complex c2[sizeh]; // components 3
} vec3_soa_t;

vec3_soa_t fermions;
```



OpenACC example for the Deo function

```
void Deo( __restrict const su3_soa * const u,
          __restrict vec3_soa * const out,
          __restrict const vec3_soa * const in,
          __restrict const double_soa * const bfield)

 int hx, y, z, t;

 #pragma acc kernels present(u) present(out) present(in) present(bfield)
 #pragma acc loop independent gang collapse(2)
 for(t=0; t<nt; t++) {
   for(z=0; z<nz; z++) {
     #pragma acc loop independent vector tile(TDIM0,TDIM1)
     for(y=0; y<ny; y++) {
       for(hx=0; hx < nxh; hx++) {

          ...
```

Nested loops over the lattice sites annotated with OpenACC directives.

Single Device Performance

Dirac Operator:

Lattice	Processor (CPU or GPU)							
	NVIDIA GK210		NVIDIA P100		Intel E5-2630v3		Intel E5-2697v4	
	SP	DP	SP	DP	SP	DP	SP	DP
24^4	4.43	8.62	1.58	2.90	70.44	94.42	51.13	66.87
32^4	4.02	9.54	1.32	2.40	79.05	100.19	43.90	54.88

Table: Measured execution time per lattice site [ns] for the Dirac operator, on several processors, in single and double precision. PGI Compiler 16.10.

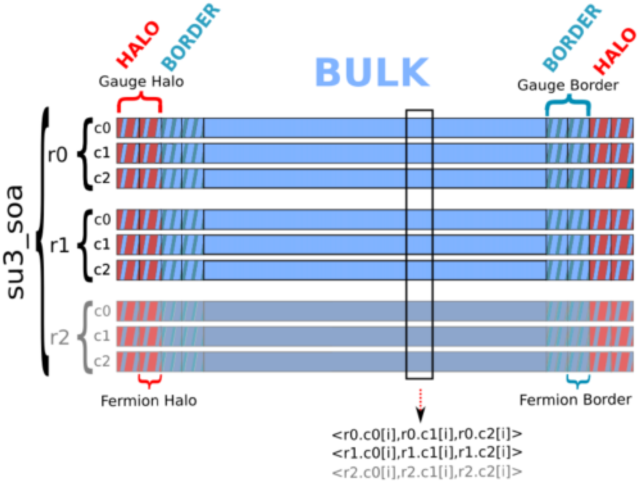
Single Device Performance

Lattice	am	β	CUDA	OpenACC	Variation
$32^3 \times 8$	0.0125	5.55	392.69	490.74	+25%
24^4	0.0125	5.55	303.80	328.07	+8%
32^4	0.001	5.52	8973.82	8228.36	-8%

Table: Execution time [sec] of a full trajectory of a complete Monte Carlo simulation for several typical physical parameters, running on one GPU of a NVIDIA K80 system.

C. Bonati, E. Calore, S. Coscetti, M. D'Elia, M. Mesiti, F. Negro, S. F. Schifano, G. Silvi, R. Tripiccione, *Design and optimization of a portable LQCD Monte Carlo code using OpenACC* International Journal Modern Physics C, 28, 1750063 (2017). doi: 10.1142/S0129183117500632

Multi Device Implementation

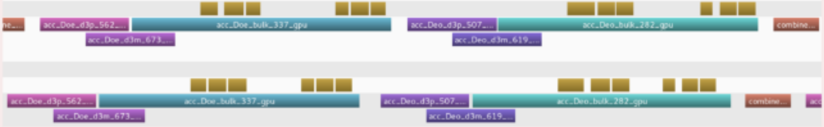


Different kernels/functions for borders and bulk operations

Overlap between computation and communication

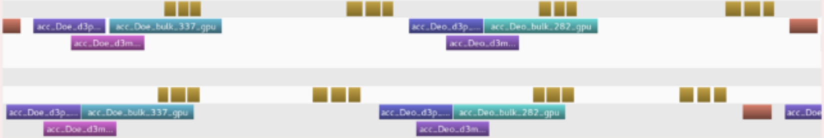
One dimensional tailing of a $32^3 \times 48$ Lattice across:

8× GPUs



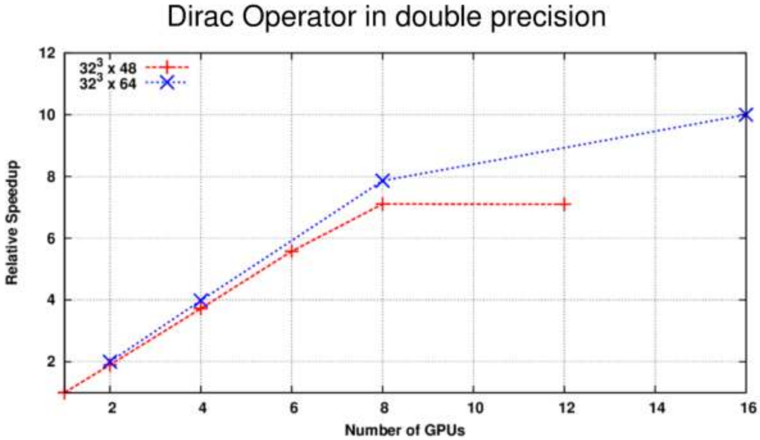
Local lattice: $32^3 \times 6$ per GPU

12× GPUs



Local lattice: $32^3 \times 4$ per GPU

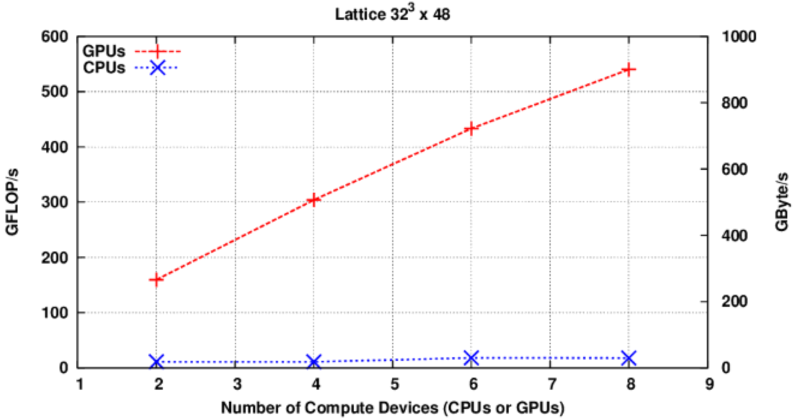
Relative Speedup on NVIDIA K80 GPUs



C. Bonati, E. Calore, M. D'Elia, M. Mesiti, F. Negro, F. Sanfilippo, S. F. Schifano, G. Silvi, R. Tripiccione, *Portable multi-node LQCD Monte Carlo simulations using OpenACC*, International Journal Modern Physics C, Accepted.

Strong Scaling Results

Roberge Weiss simulation over a $32^3 \times 48$ lattice, with mass 0.0015 and beta 3.3600, using mixed precision floating-point.



Using 2 CPUs we measure a $\approx 14\times$ increase in the execution time wrt using 2 GPUs and the gap widens for more devices.

Conclusions

LQCD Monte Carlo

- a single code version able to run on different architectures
- capability to run on several computing devices / nodes
- still regular plain C code if ignoring directives
- performance comparable to CUDA implementations on NVIDIA GPUs

Future works

- investigate performance of multi-dimensional tiling
- experiment different compilers targeting Intel CPUs (e.g. GCC 7)
- introduce optimizations for Intel CPUs and MICs without impacting GPU performance
- study energy-saving strategies

Thanks for Your attention