CUDA, OpenCL and OpenACC experiences for Lattice Boltzmann simulations

Enrico Calore, Sebastiano Fabio Schifano, Raffaele Tripiccione

University of Ferrara and INFN-Ferrara

SUMA meeting

April 1, 2013

Ferrara, ITALIA

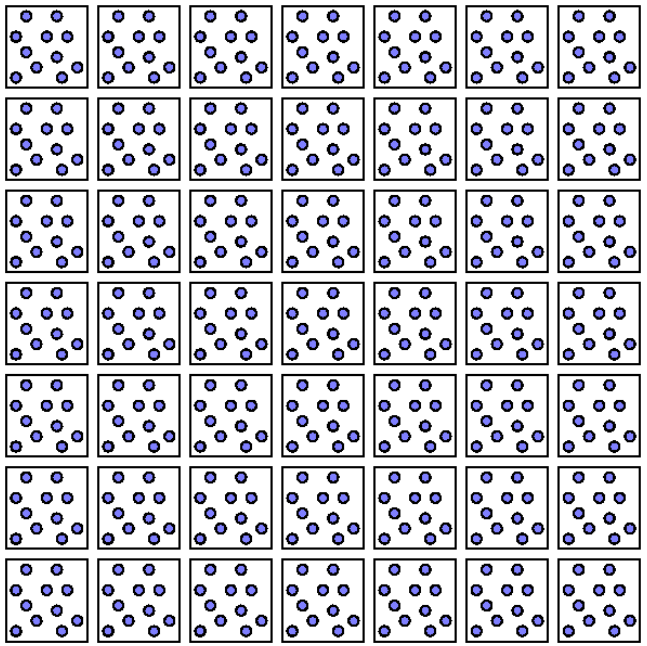
The D2Q37 Lattice Boltzmann Model

- 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
```



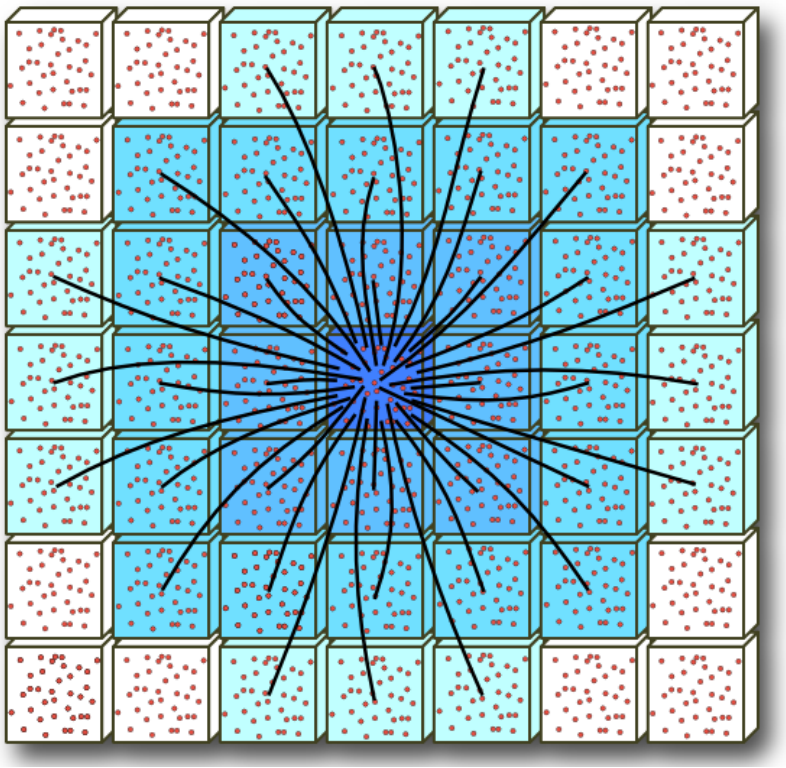
Embarassing parallelism

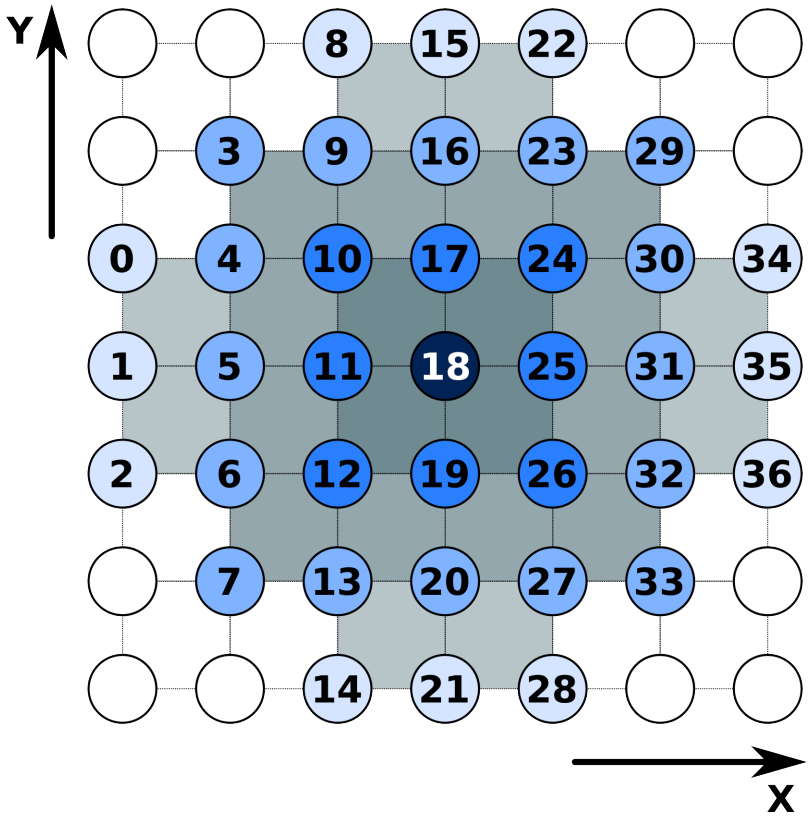
All sites can be processed in parallel applying in sequence propagate and collide.

Challenge

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

D2Q37: propagation scheme

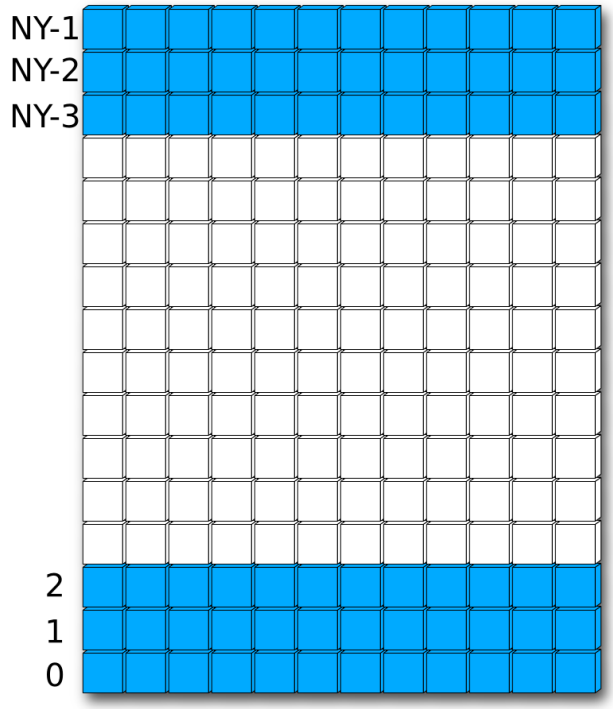




- require to access neighbours cells at distance 1,2, and 3,
- generate memory-accesses with **sparse** addressing patterns.

D2Q37: boundary-conditions

- we simulate a 2D lattice with period-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

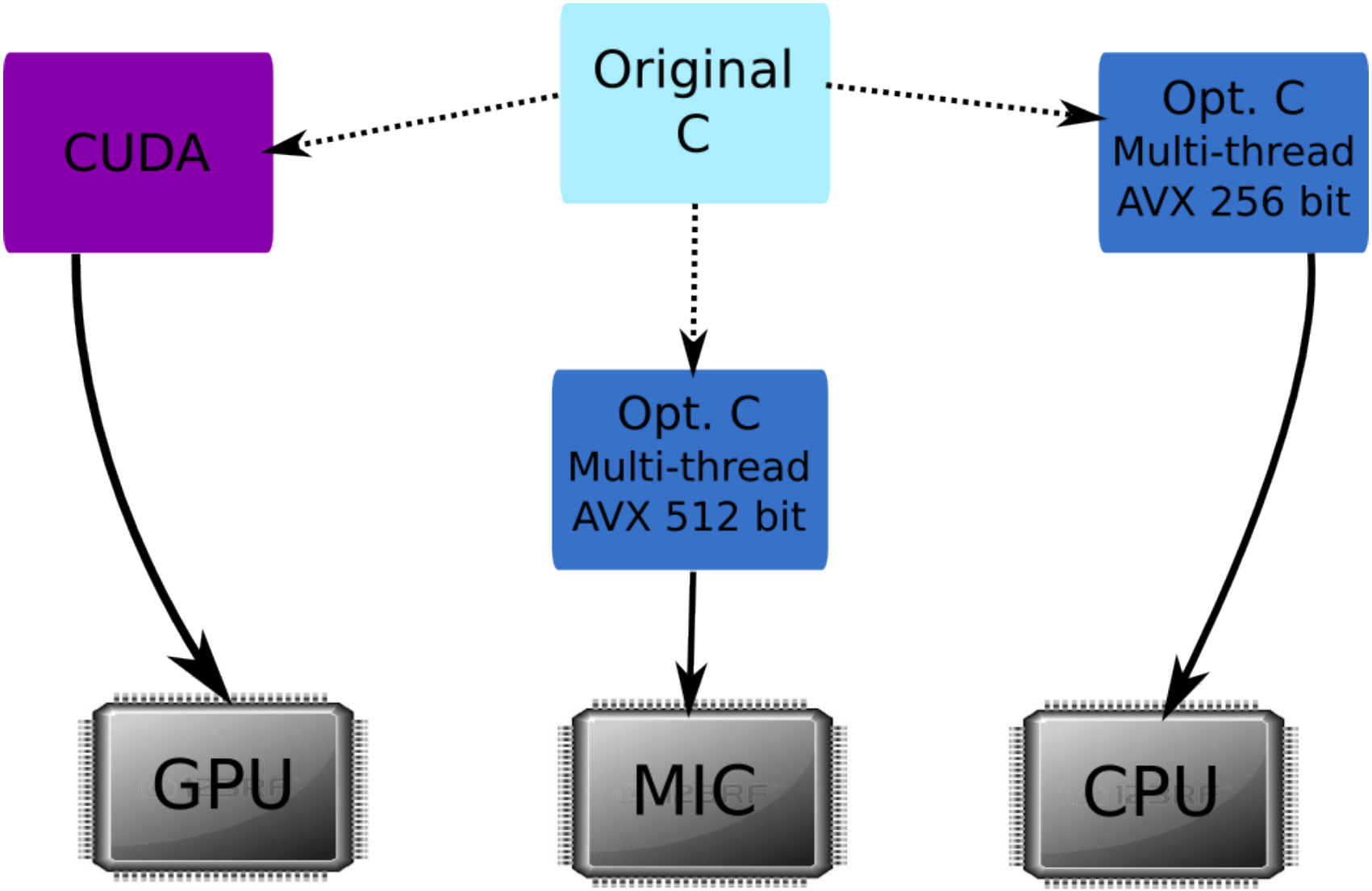


This step (bc) is computed before the collision step.

D2Q37 collision

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

Code implementations



Memory layout for LB : AoS vs SoA

- lattice stored as AoS:

```
typedef struct {
  double  p1; // population 1
  double  p2; // population 2
  ...
  double p37; // population 37
} pop_t;

pop_t lattice2D[SIZEX*SIZEY];
```

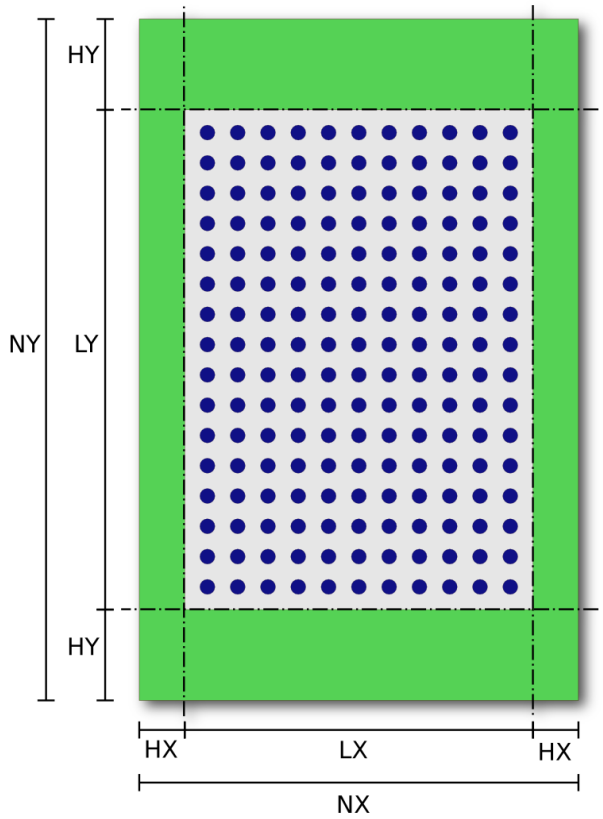
- lattice stored as SoA:

```
typedef struct {
  double  p1[SIZEX*SIZEY]; // population 1 array
  double  p2[SIZEX*SIZEY]; // population 2 array
  ...
  double p37[SIZEX*SIZEY]; // population 37 array
} pop_t;

pop_t lattice2D;
```

- AoS suitable for CPU: improves cache-locality for computing collision
- SoA suitable for GPU: improves data coalescing

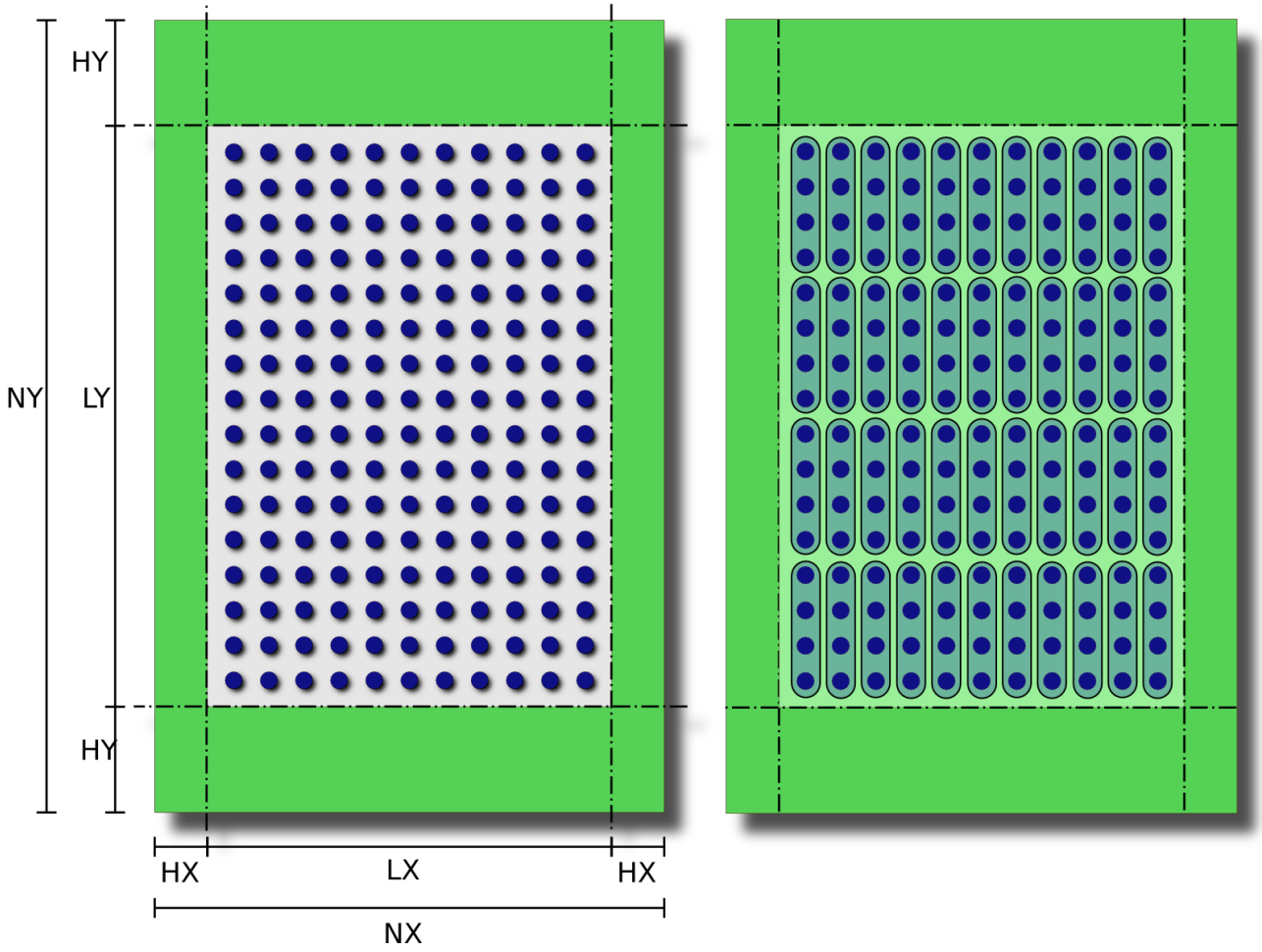
Lattice memory allocation

- lattice is allocated in column-major order
- on GPU we allocated two copies of the lattice:
  each step reads from **prv** and write onto **nxt**
- a lattice of size $L_x \times L_y$ is allocated as a grid of $(3 + L_x + 3) \times (16 + L_y + 16)$ sites:
  - make uniform computation of propagate also for sites close to borders
  - along y-direction the size is warp- (32) and cache-aligned (128B)



GPU Grid Layouts

For all GPU kernels *blocks*/*work-group*/*gangs* are configured as 1D array of *threads*/*work-items*/*vectors*, each processing one lattice site.



Example of a physical lattice of 11 $\times$ 16 cells.

Towards an hardware independent code

- OpenCL
  - Framework for writing programs that execute across heterogeneous platforms (CPUs, GPUs, MICs, FPGAs, etc.)
  - Open standard developed by the not-for-profit Khronos group, supported by Apple, Intel, AMD, (NVIDIA), etc.
  - Apparently NVIDIA do not support it anymore
- OpenACC
  - Directive based programming standard for heterogeneous parallel computing
  - Developed by Cray, CAPS, Nvidia and PGI
  - At the moment it addresses only NVIDIA GPUs and some AMD GPUs

Independence may have costs in terms of complexity and performance

Towards an hardware independent code

- OpenCL
  - Framework for writing programs that execute across heterogeneous platforms (CPUs, GPUs, MICs, FPGAs, etc.)
  - Open standard developed by the not-for-profit Khronos group, supported by Apple, Intel, AMD, (NVIDIA), etc.
  - Apparently NVIDIA do not support it anymore
- OpenACC
  - Directive based programming standard for heterogeneous parallel computing
  - Developed by Cray, CAPS, Nvidia and PGI
  - At the moment it addresses only NVIDIA GPUs and some AMD GPUs

Independence may have costs in terms of complexity and performance

OpenCL/“CUDA” example

Propagate device function:

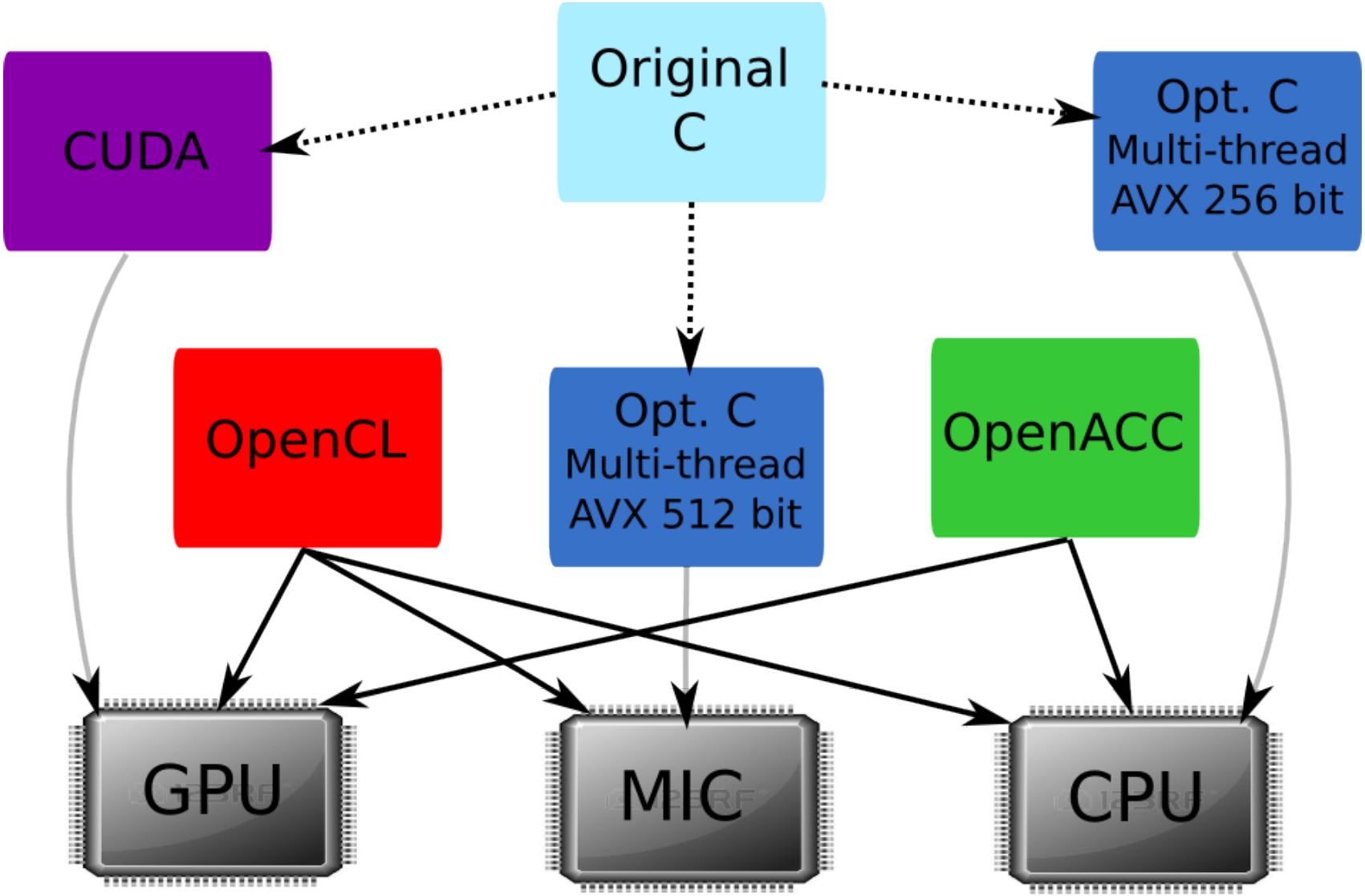
```
__kernel void propagate(__global const data_t* prv, __global data_t* nxt) {

  int ix,           // Work-item index along the X dimension.
      iy,           // Work-item index along the Y dimension.
      site_i;       // Index of current site.

  // Sets the work-item indices (Y is used as the fastest dimension).
  ix = (int) get_global_id(1);
  iy = (int) get_global_id(0);

  site_i = (HX+3+ix)*NY + (HY+iy);

  nxt[           site_i] = prv[           site_i - 3*NY + 1];
  nxt[   NX*NY + site_i] = prv[   NX*NY + site_i - 3*NY    ];
  nxt[ 2*NX*NY + site_i] = prv[ 2*NX*NY + site_i - 3*NY - 1];
  nxt[ 3*NX*NY + site_i] = prv[ 3*NX*NY + site_i - 2*NY + 2];
  nxt[ 4*NX*NY + site_i] = prv[ 4*NX*NY + site_i - 2*NY + 1];
  nxt[ 5*NX*NY + site_i] = prv[ 5*NX*NY + site_i - 2*NY    ];
  nxt[ 6*NX*NY + site_i] = prv[ 6*NX*NY + site_i - 2*NY - 1];

  ...
```

OpenACC example

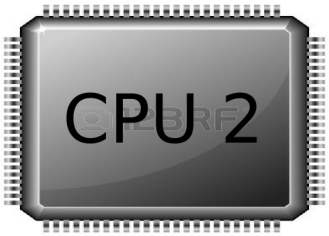
Propagate function:

```
inline void propagate(const data_t* restrict prv, data_t* restrict nxt ) {

  int ix, iy, site_i;

  #pragma acc kernels present(prv) present(nxt)
  #pragma acc loop gang independent
  for ( ix=HX; ix < (HX+SIZEX); ix++) {
    #pragma acc loop vector independent
    for ( iy=HY; iy<(HY+SIZEY); iy++) {

      site_i = (ix*NY) + iy;

      nxt[           site_i] = prv[           site_i - 3*NY + 1];
      nxt[   NX*NY + site_i] = prv[   NX*NY + site_i - 3*NY    ];
      nxt[ 2*NX*NY + site_i] = prv[ 2*NX*NY + site_i - 3*NY - 1];
      nxt[ 3*NX*NY + site_i] = prv[ 3*NX*NY + site_i - 2*NY + 2];
      nxt[ 4*NX*NY + site_i] = prv[ 4*NX*NY + site_i - 2*NY + 1];
      nxt[ 5*NX*NY + site_i] = prv[ 5*NX*NY + site_i - 2*NY    ];
      nxt[ 6*NX*NY + site_i] = prv[ 6*NX*NY + site_i - 2*NY - 1];

 ...
```

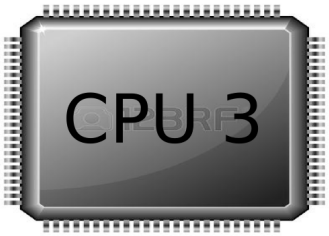
Code implementations



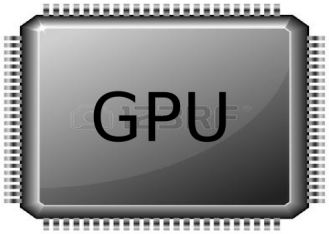
Used processors/accelerators (Eurora)



2 x Intel Xeon Processor E5-2658 2.10 GHz (8 core)



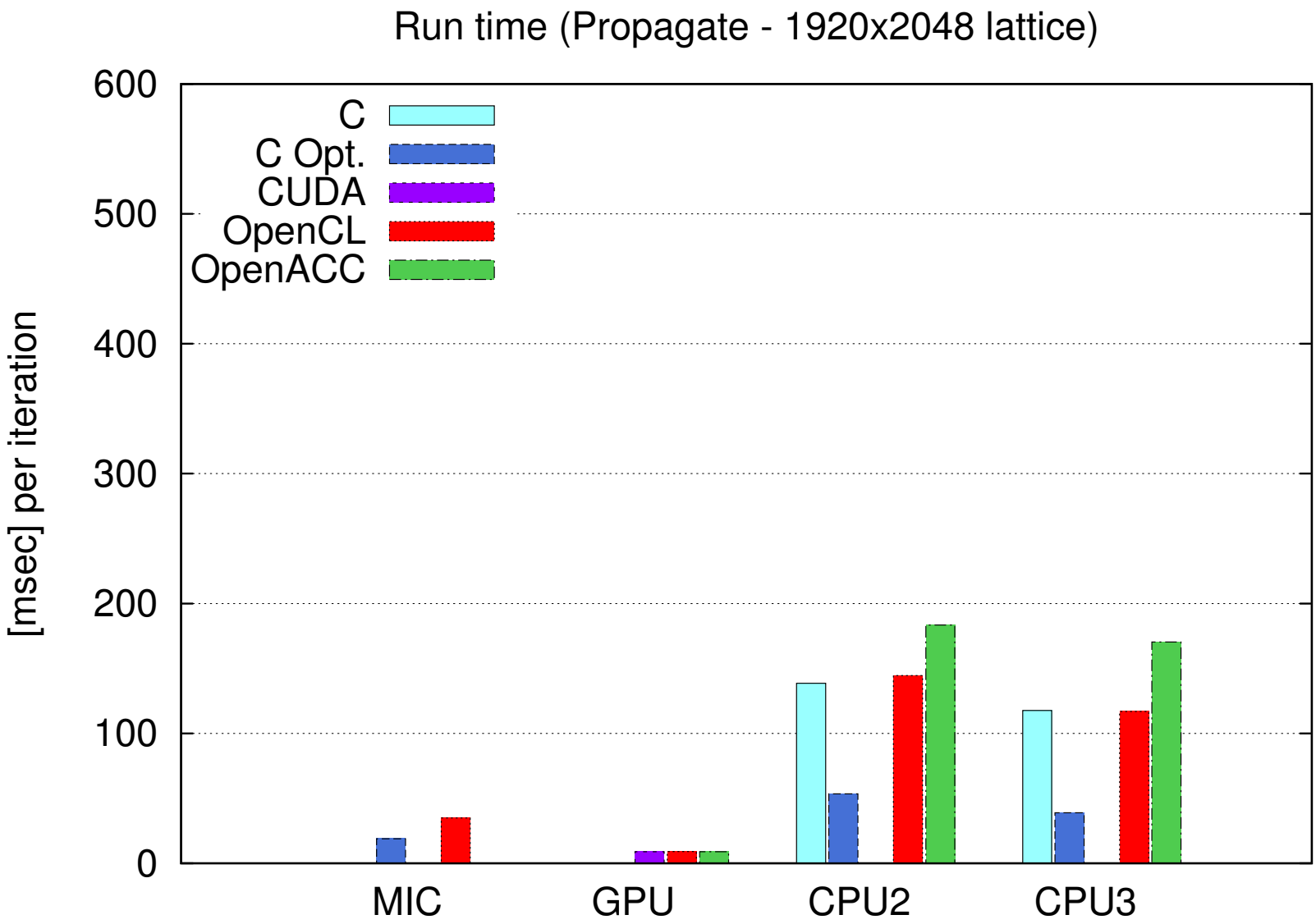
2 x Intel Xeon Processor E5-2687W 3.10 GHz (8 core)



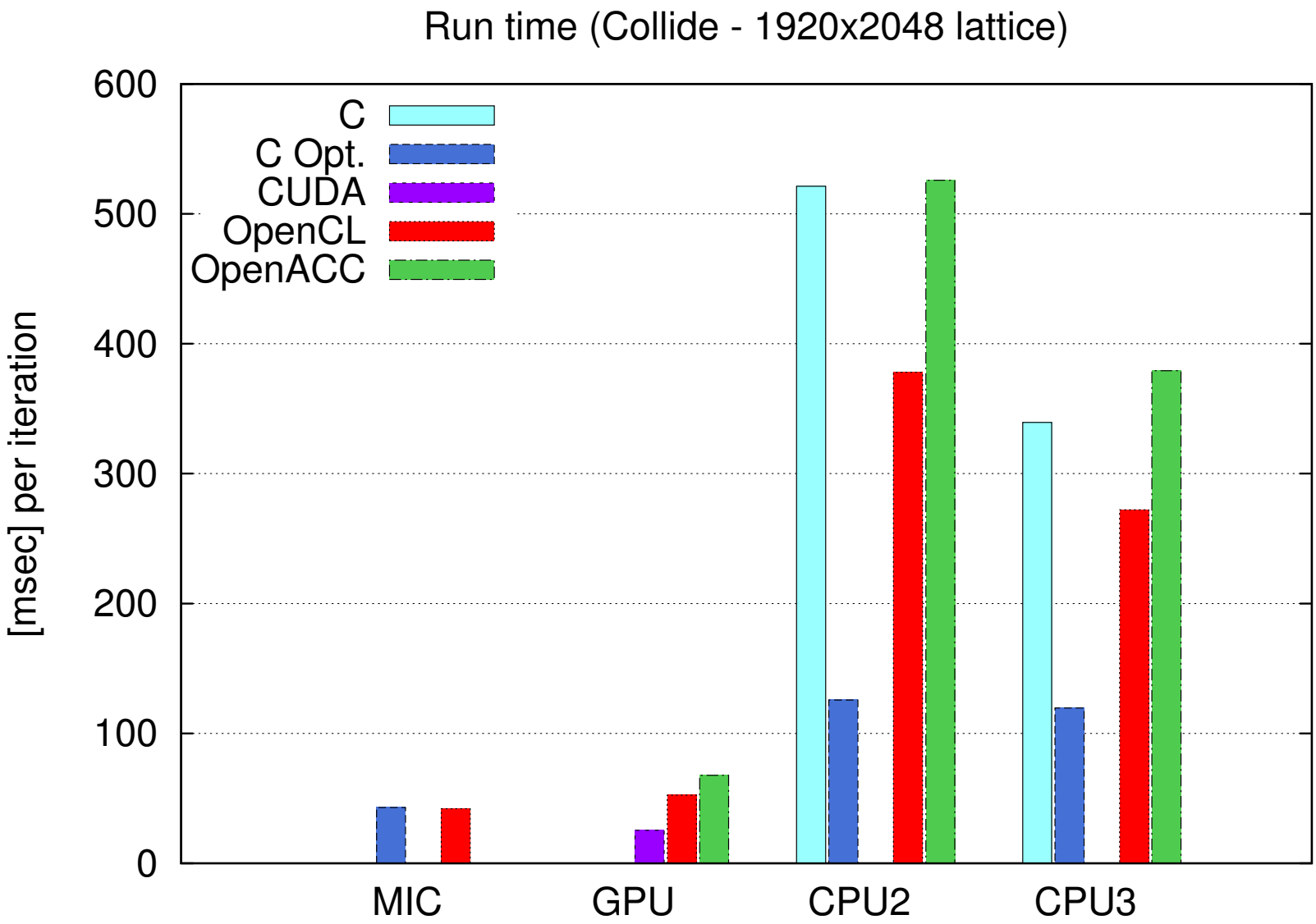
2 x NVIDIA Tesla K20s (Kepler cc 3.5)

2 x Intel Xeon-Phi 5120D 1.053GHz

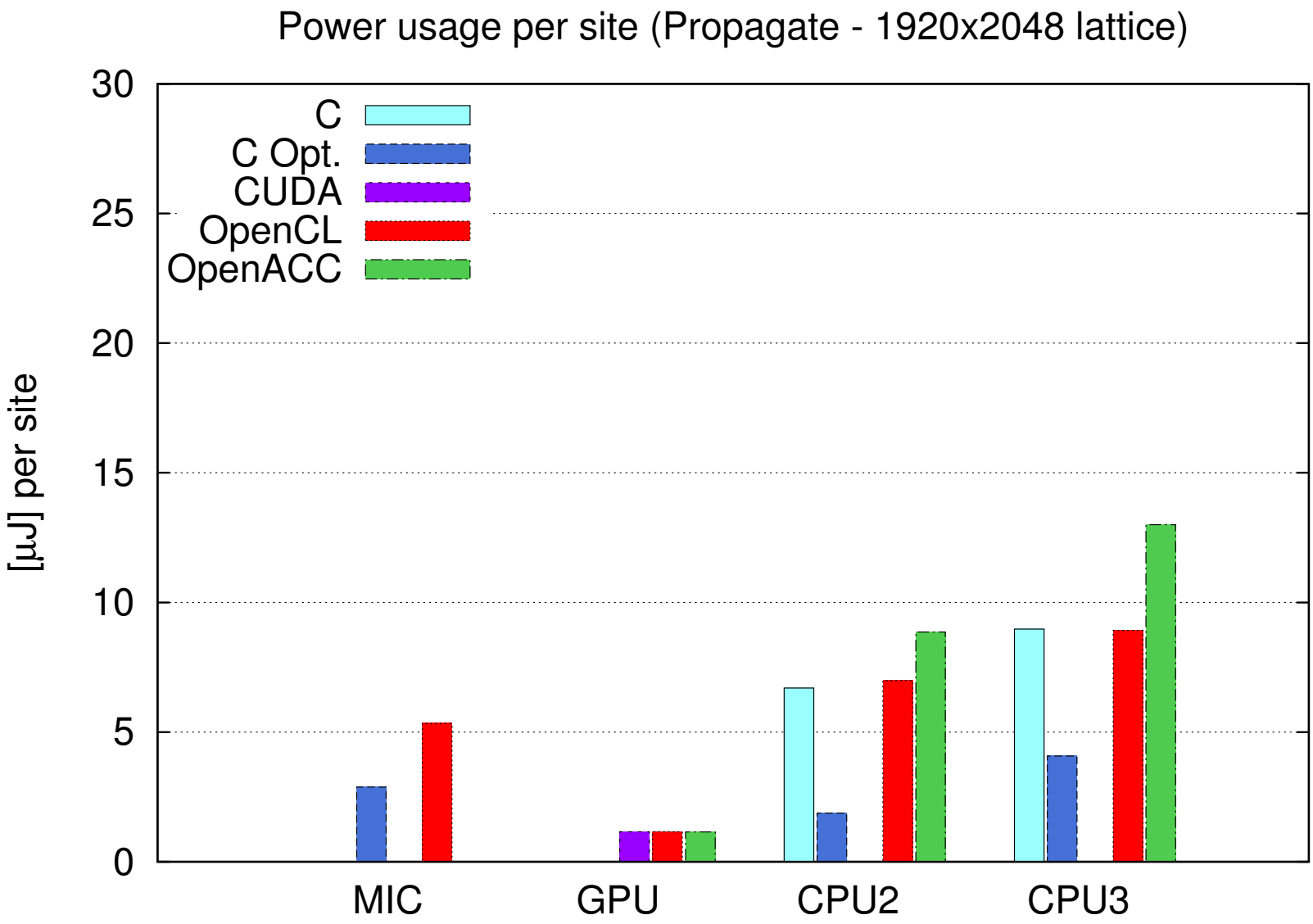
Run Time Comparison



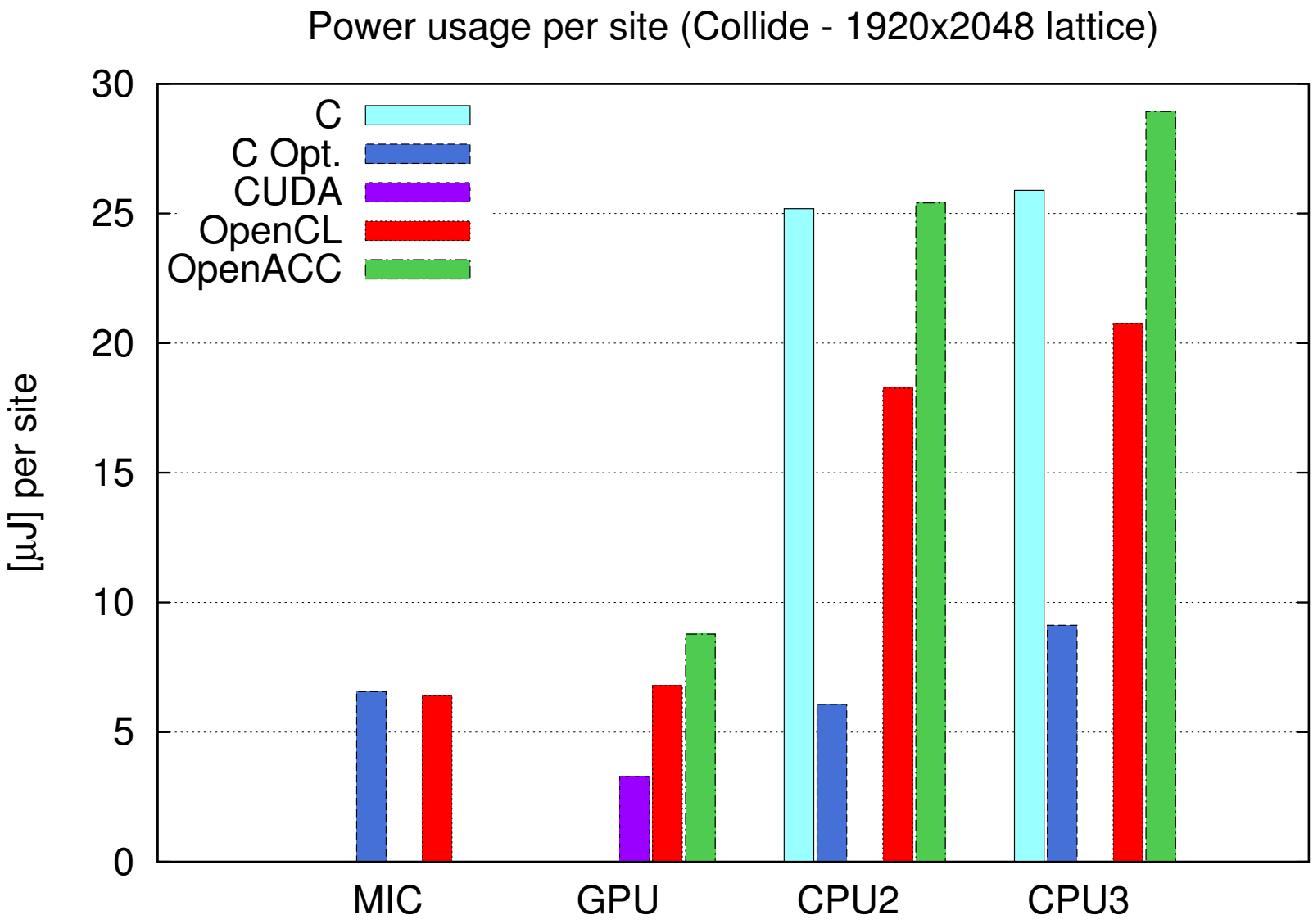
Run Time Comparison



Energy Efficiency Comparison



Energy Efficiency Comparison



Thanks for Your attention