Improving 3D Lattice Boltzmann Method with asynchronous transfers on many-core processors

Minh Quan HO [1,3], Bernard TOURANCHEAU [1], Christian OBRECHT [2], Benoît DUPONT DE DINECHIN [3] and Julien HASCOET [3]

[1]LIG UMR 5217 - Grenoble Alps University - Grenoble, France
[2]CETHIL UMR 5008 - INSA-Lyon - Villeurbanne, France
[3]Kalray S.A. - Montbonnot, France

CCDSC - October 03-06, 2016

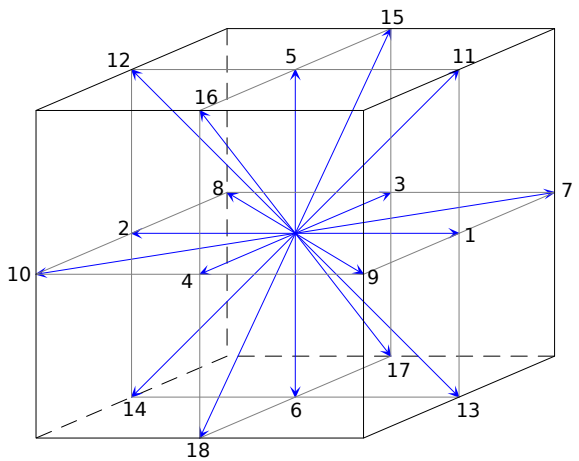
Overview

Introduction - LBM theory

The Lattice Boltzmann Method performs on a regular Cartesian grid:

- mesh size δx
- constant time step δt
- A node $= \{$particle densities f_α, velocities $\xi_\alpha\}$
- Nodes are linked by e.g the D3Q19 stencil and updated by [He, 1997]:

$$\left| f_\alpha(x + \delta t \xi_\alpha, t + \delta t) \right\rangle - \left| f_\alpha(x, t) \right\rangle = \Omega \left(\left| f_\alpha(x, t) \right\rangle \right) \tag{1}$$



Introduction - Memory bound context

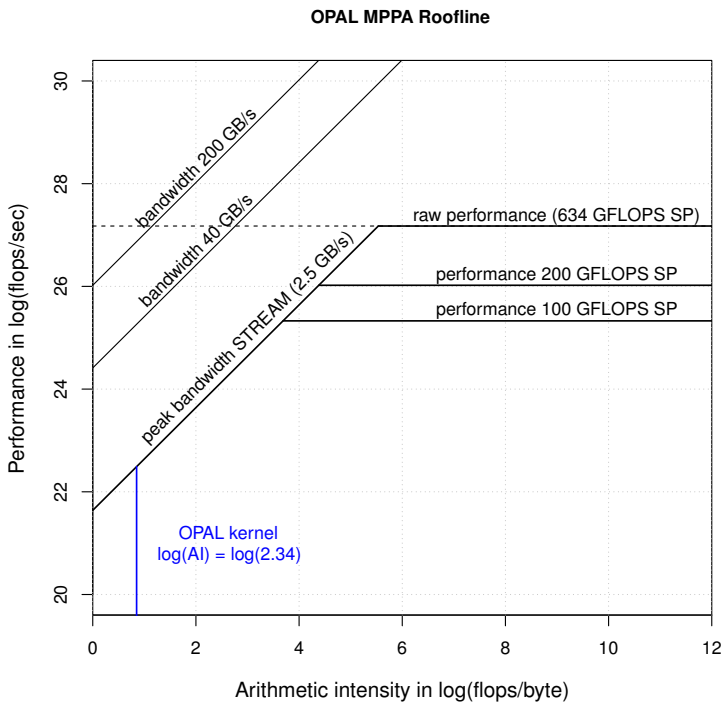
Given a 'square' fluid represented as a grid of $L \times L \times L$ lattice nodes in D3Q19, evoluating throught T time steps.

Simulating the whole domain requires:

- $19 \times 2 \times L^3 \times T$ floating-point numbers
- $\leq 400 \times L^3 \times T$ floating-point ops.

Moving data is much slower than computing today.

GPU is until now the most well-suited for LBM.



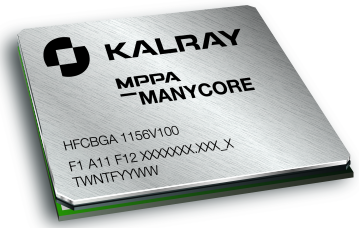
Motivation

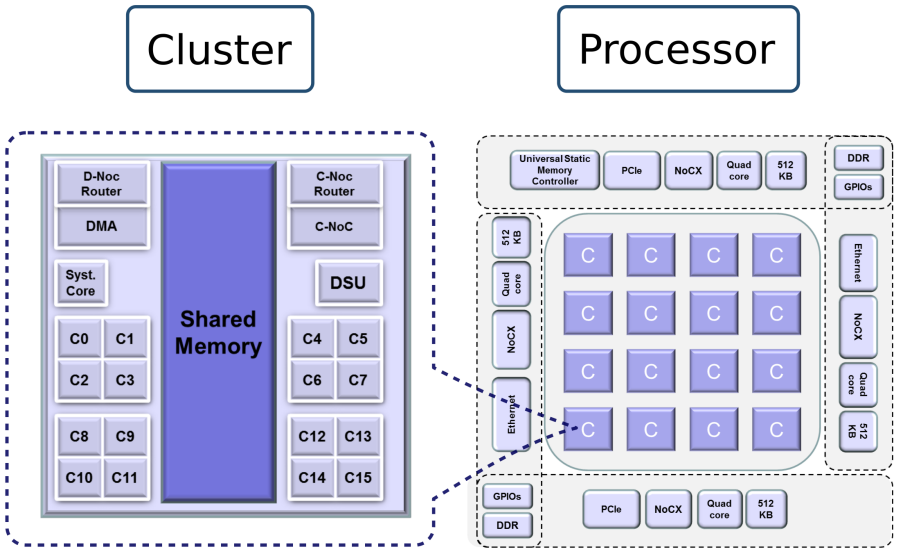
- Power-efficient NoC-based many-core processors are very promising for next HPC challenges (e.g Sunway, MPPA, PULP, STHORM ...).
- Good latency, but low memory bandwidth (DDR3).
- Lack of efficient programming model and optimization methods.
- High {computing|data} predictability and fast-local-memory centric.
- Enabling sophisticated optimizations, based on software-prefetching and streaming.

These motivates us to study a pipelined 3D LBM algorithm on many-core processors, using local memory and asynchronous communication.

Kalray MPPA-256 architecture

- 16 x 16-core Compute Clusters (CC)
- 2 x I/O clusters with quad-core CPUs, DDR3, Ethernet, PCIe
- Dual 2D-torus NoC for 24 GB/s per link @ 600 MHz
- Peak 634 GFLOPS SP for 25W @ 600 MHz

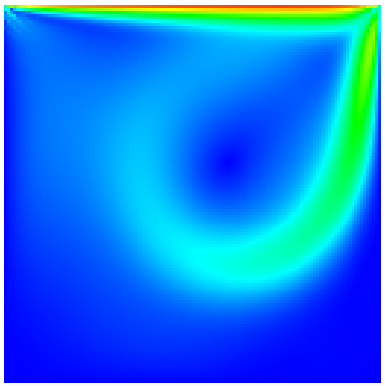


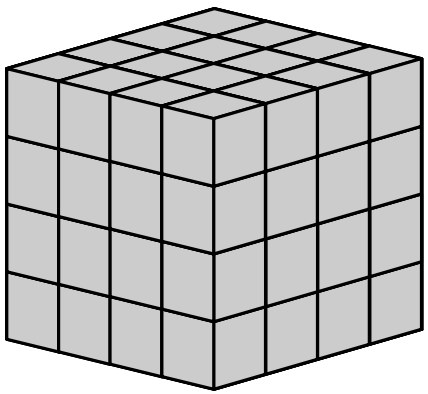


- 2 MB multi-banked shared memory per CC, 77 GB/s bandwidth
- SMEM configurable as DDR L2 cache, or explicit user buffers
- Support asynchronous data transfer by DMA engines
- POSIX C/C++ programming or OpenCL offloading

Outline

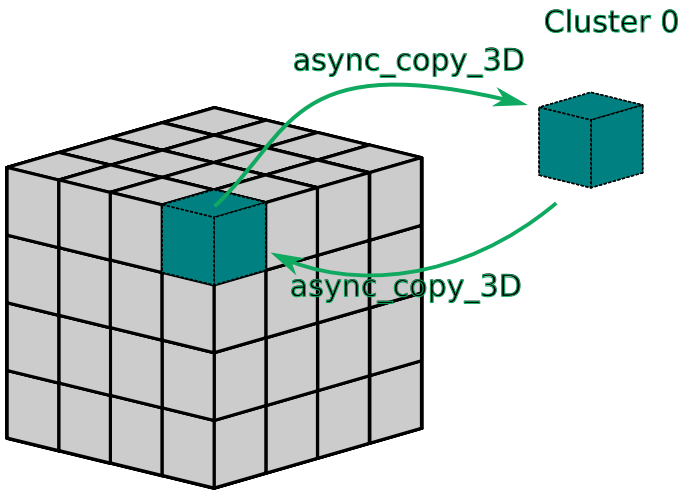
Domain decomposition and macro pipeline





- We take the *lid-driven cavity* example from the OPAL solver [Obrecht, 2015], originally implemented in OpenCL
- The $L_x \times L_y \times L_z$ domain is decomposed to sub-domains of size $C_x \times C_y \times C_z$

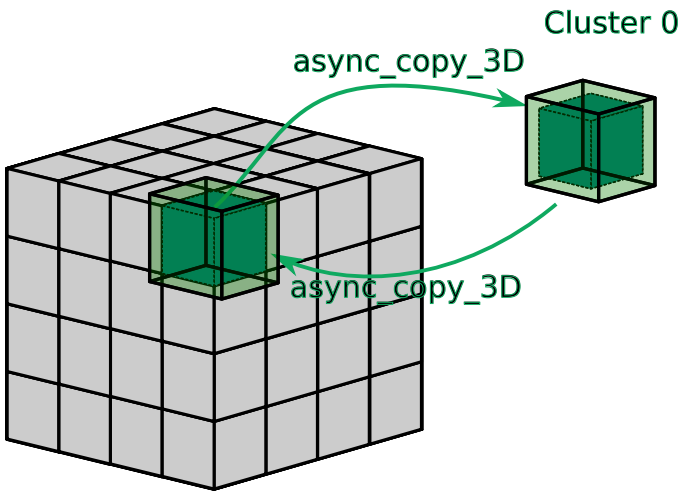
Domain decomposition and macro pipeline



Main idea:

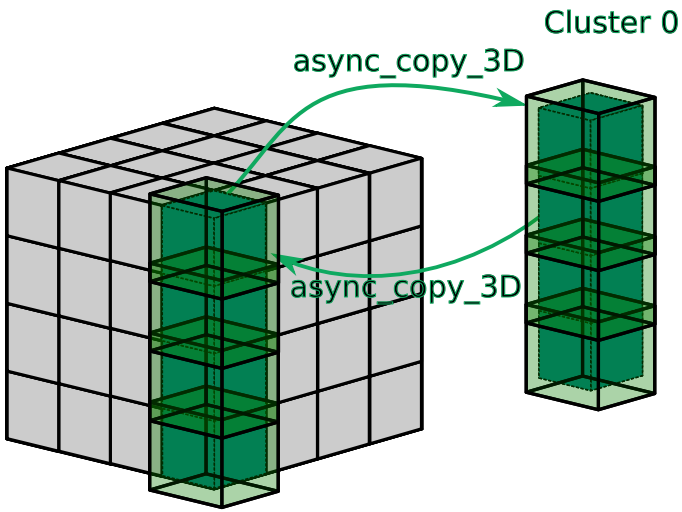
- A sub-domain is copied into CC's local memory by a 3D asynchronous copy function
- Computation is carried out on local memory then data are copied back to global memory (DDR)

Domain decomposition and macro pipeline



- Requires copying halo layers for each sub-domain
- In 1-order stencil, the copied sub-domain S is at most $(C_x + 2) \times (C_y + 2) \times (C_z + 2)$

Domain decomposition and macro pipeline



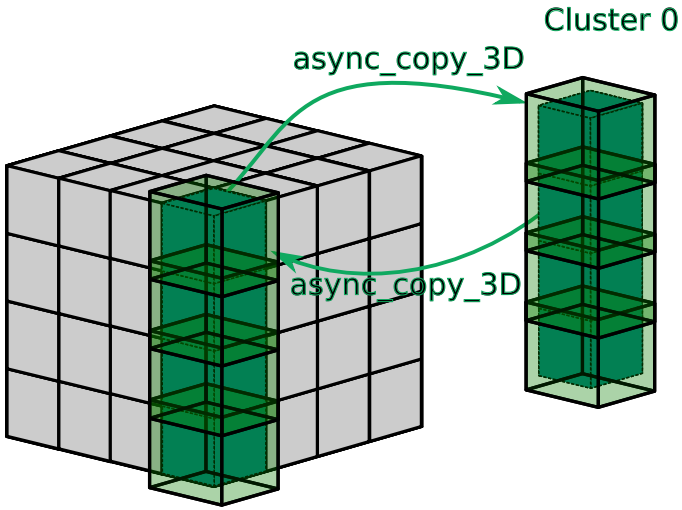
16 computing clusters, each is working on NB_CUBES_PER_CLUSTER sub-domains:

```
/* Prologue */
prefetch_cube(0); // non-blocking

/* Pipeline */
for i in 0 .. NB_CUBES_PER_CLUSTER-1
  prefetch_cube(i+1); // non-blocking
  wait_cube(i);
  compute_cube(i);
  put_cube(i);
done

/* Epilogue */
wait_cube(NB_CUBES_PER_CLUSTER-1);
```

Outline

Sub-domain addressing

A : "Hey, don't touch my cube !"
B : "No, that's mine."

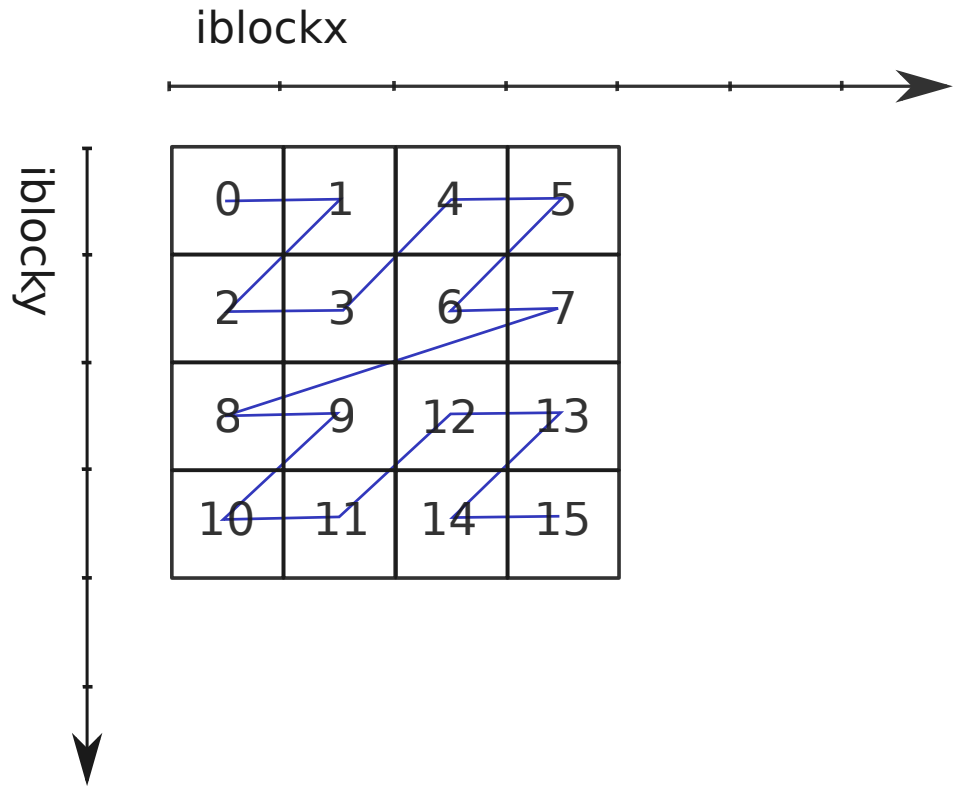


[a]

[a] Credit : 9gag

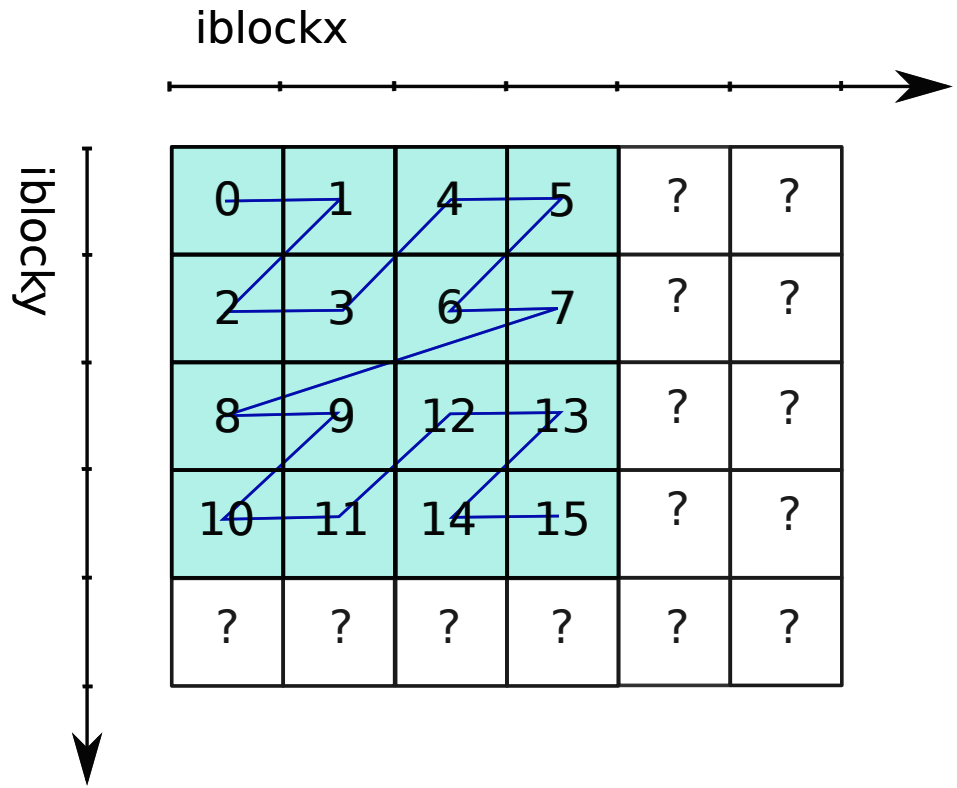
Sub-domain addressing

- Space filling curves like Morton or Hilbert are fast



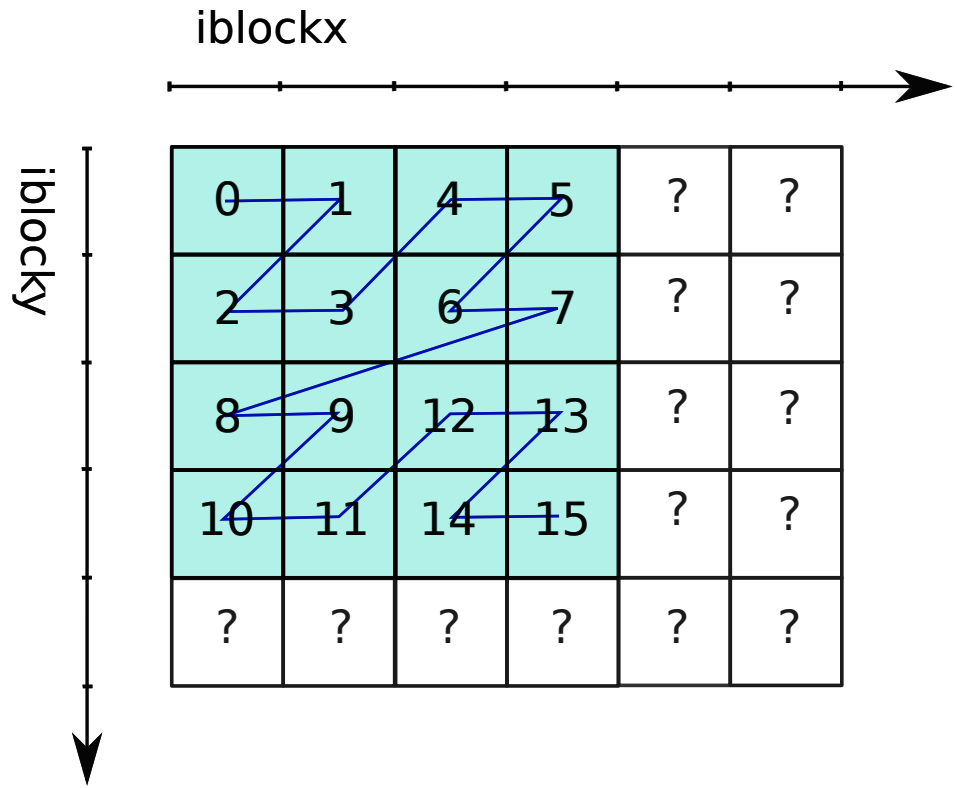
Sub-domain addressing

- Space filling curves like Morton or Hilbert are fast
- But, what if (sub-)domains are not cubic ?



Sub-domain addressing

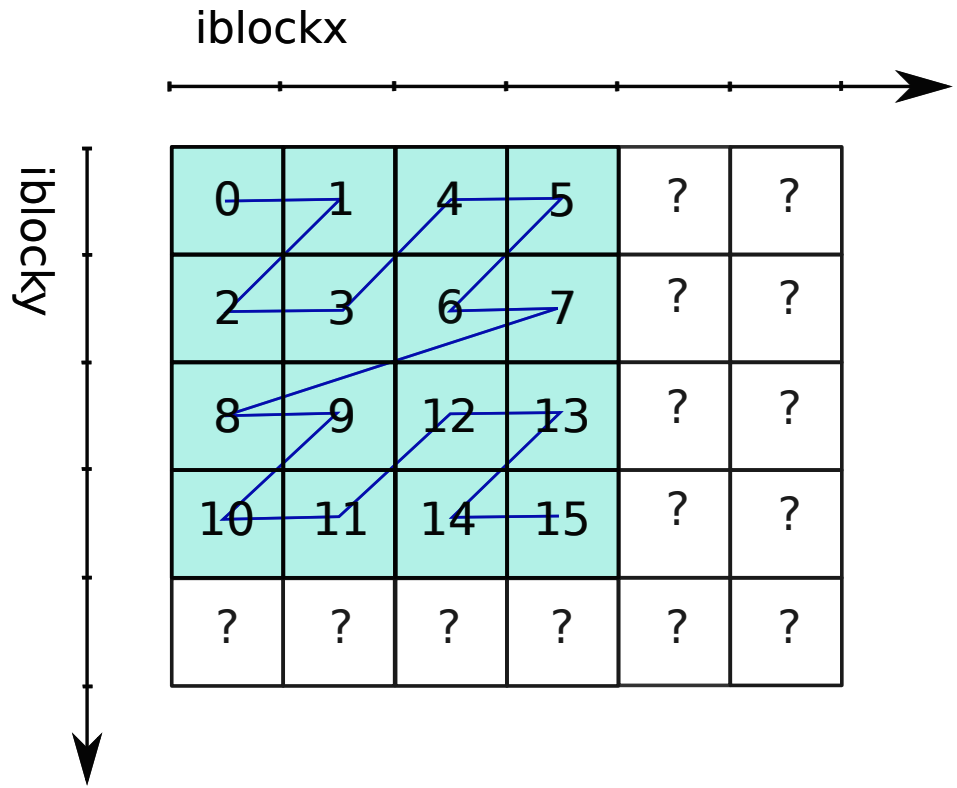
- Space filling curves like Morton or Hilbert are fast
- But, what if (sub-)domains are not cubic ?



- Such a curve that works for any configuration will be more complex (octree, recursion, trailing handle)

Sub-domain addressing

- Space filling curves like Morton or Hilbert are fast
- But, what if (sub-)domains are not cubic ?

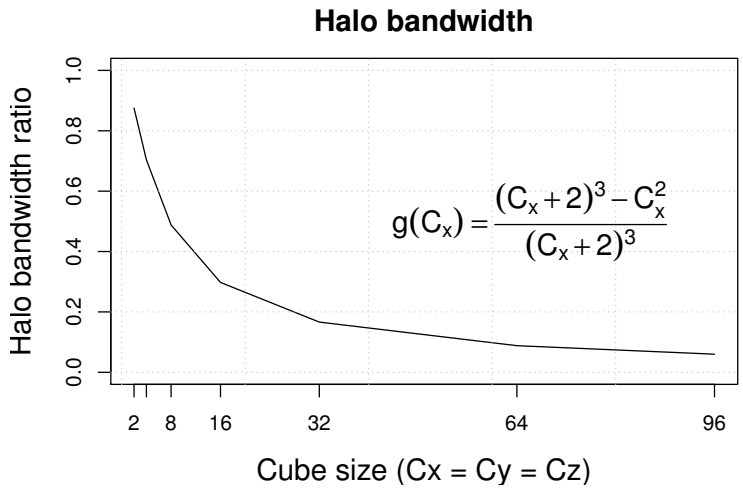


- Such a curve that works for any configuration will be more complex (octree, recursion, trailing handle)
- **Addressing sub-domains in '3D' row-major style**

Outline

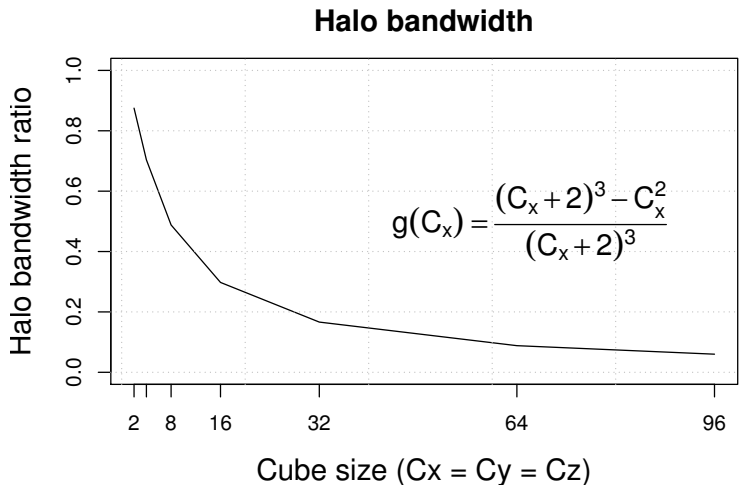
Sub-domain size and Halo bandwidth

- We call "Halo bandwidth" the fraction between the number of halo cells and the total number of copied cells.



Sub-domain size and Halo bandwidth

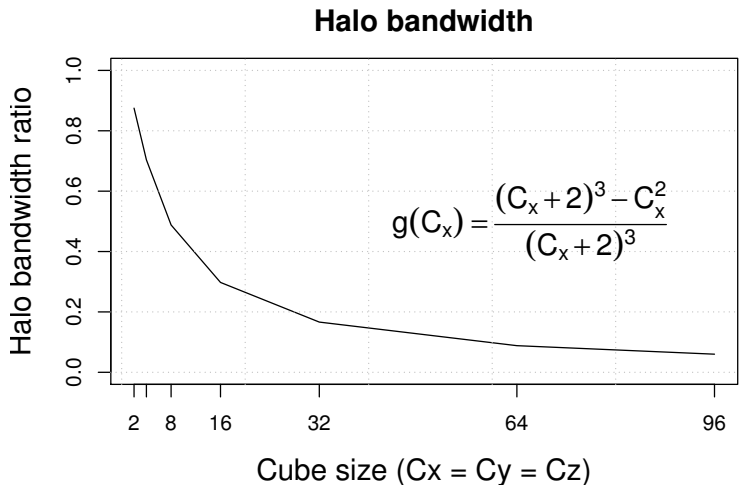
- We call "Halo bandwidth" the fraction between the number of halo cells and the total number of copied cells.



- Which size for sub-domains, given a limited local memory ?

Sub-domain size and Halo bandwidth

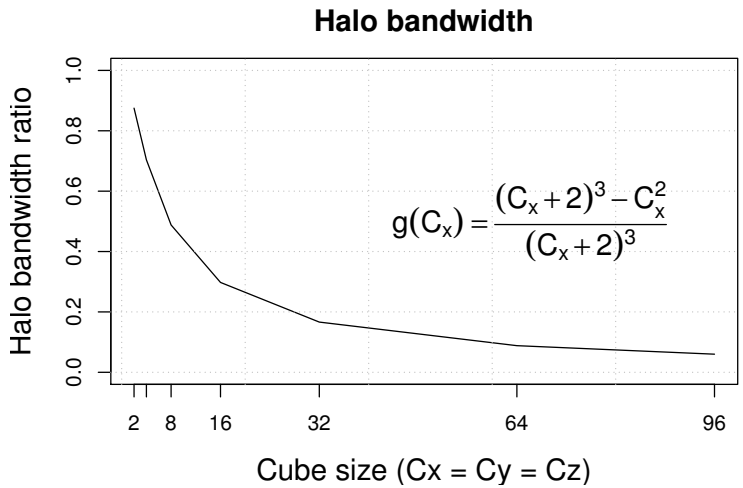
- We call "Halo bandwidth" the fraction between the number of halo cells and the total number of copied cells.



- Which size for sub-domains, given a limited local memory ?
- E.g double buffering :
  $\texttt{malloc}(2 \times (C_x + 2)^3 \times \mathit{sizeof}(\mathit{float}))$ $(C_x = C_y = C_z)$

Sub-domain size and Halo bandwidth

- We call "Halo bandwidth" the fraction between the number of halo cells and the total number of copied cells.

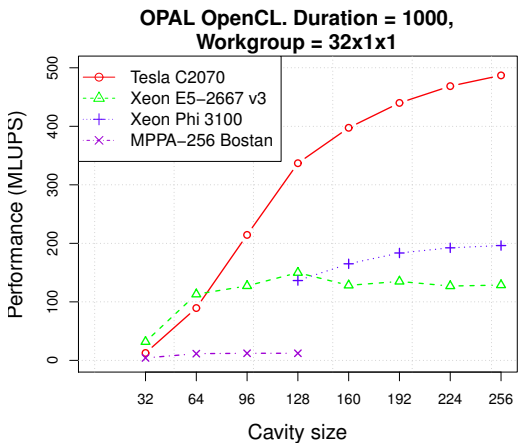


- Which size for sub-domains, given a limited local memory ?
- E.g double buffering :
  $\texttt{malloc}(2 \times (C_x + 2)^3 \times \mathit{sizeof}(\mathit{float}))$ $(C_x = C_y = C_z)$
- Sub-domains should be cubic and as big as possible.

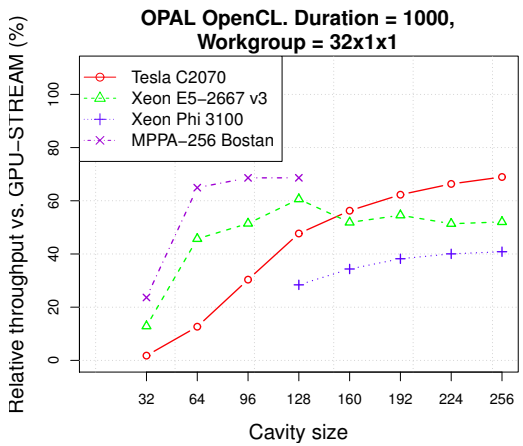
Outline

Results (1/2)

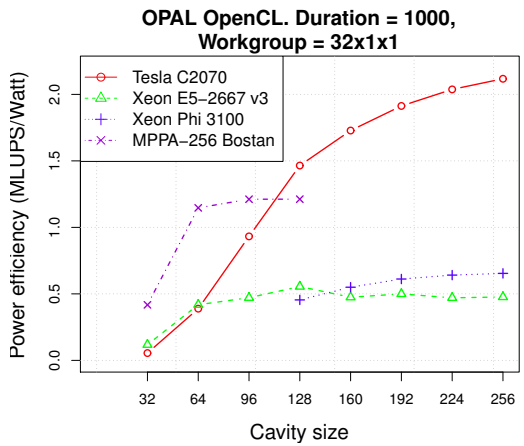
We compare original OPAL performance on Intel CPU, Intel MIC, NVIDIA GPU and Kalray MPPA-256 (all OpenCL).



(a) Performance in MLUPS



(b) Relative throughput to GPU-STREAM (%)



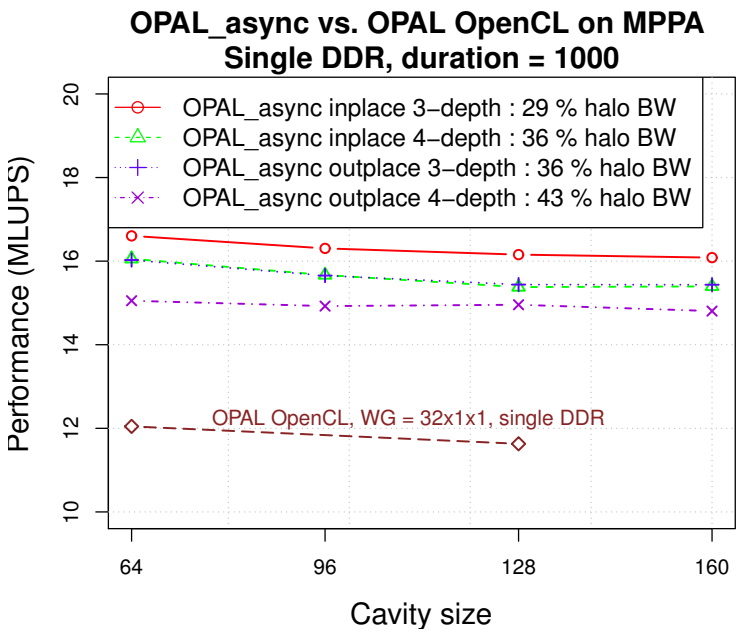
(c) Power efficiency (MLUPS/W)

Figure: Original OPAL OpenCL on GPU, CPU, MIC and MPPA

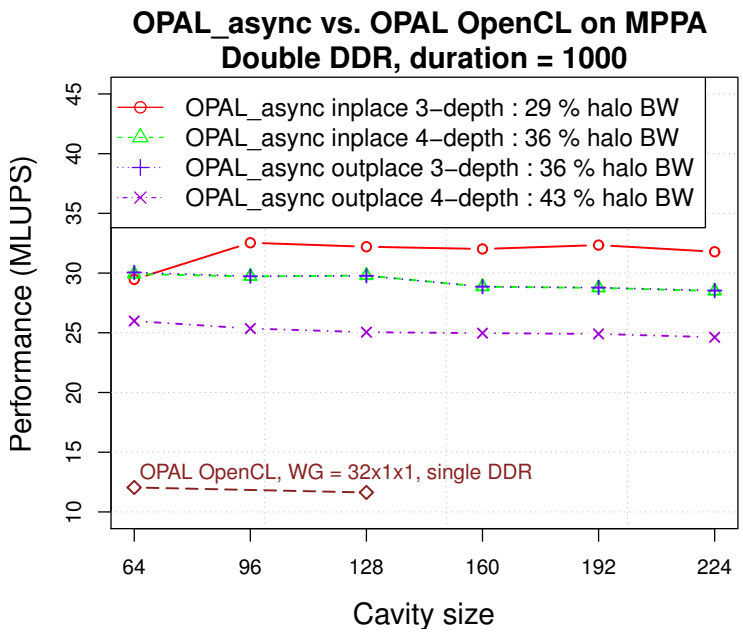
GPU-STREAM benchmark [Deakin, 2015]

Results (2/2)

- Asynchronous approach implemented in POSIX C on MPPA
- Outperforms the OpenCL version by 33%
- Twice better using two DDRs (MPPA OpenCL currently supports only one DDR)



(a) Single-DDR



(b) Double-DDR

Figure: OPAL_async vs. OPAL OpenCL on MPPA

Conclusions

- 33% performance gain by actively streaming stencil domains on local memories.
- Software pipeline is not a trivial task, but essential to obtain good performance on many-core processors.
- DDR bandwidth is bottleneck.
- Halo copy is critical to performance, consumes up to 60% bandwidth on small sub-domains.
- Perspective : applying alternative method - Link-wise artificial compressibility method [Obrecht, 2016] with 5x less memory traffic.

References

He, Xiaoyi, and Li-Shi Luo (1997)

Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation.

Physical Review E 56.6 (1997): 6811.

Obrecht, Christian, Bernard Tourancheau, and Frdric Kuznik (2015)

Performance Evaluation of an OpenCL Implementation of the Lattice Boltzmann Method on the Intel Xeon Phi.

Parallel Processing Letters 25.03 (2015): 1541001.

Deakin, Tom, and Simon McIntosh-Smith (2015)

GPU-STREAM: Benchmarking the achievable memory bandwidth of Graphics Processing Units.

Supercomputing Poster Austin, Texas (2015).

Obrecht, Christian, et al. (2016)

Thermal link-wise artificial compressibility method: GPU implementation and validation of a double-population model.

Computers & Mathematics with Applications 72.2 (2016): 375-385.