An OpenCL simulation of molecular dynamics on heterogeneous architectures

Master's thesis

Samuel Pitoiset

Internship supervisor : Raymond NAMYST

LaBRI/INRIA - RUNTIME team

October 9, 2014

Summary

Short range N-body simulation

Molecular dynamics (MD)

- computer simulation of a system of particles;
- N-body problem (cut-off distance):
  - forces are neglected if $dist(part1, part2) > r_c$.

Motivation

- simulate hundreds of millions of particles;
- verify simulation results with real experiments (physicist).

Goals

- use multiple accelerators on a single node;
- integrate the simulation to ExaStamp (CEA):
  - a parallel framework for MD on heterogeneous clusters.

Overview of the simulation

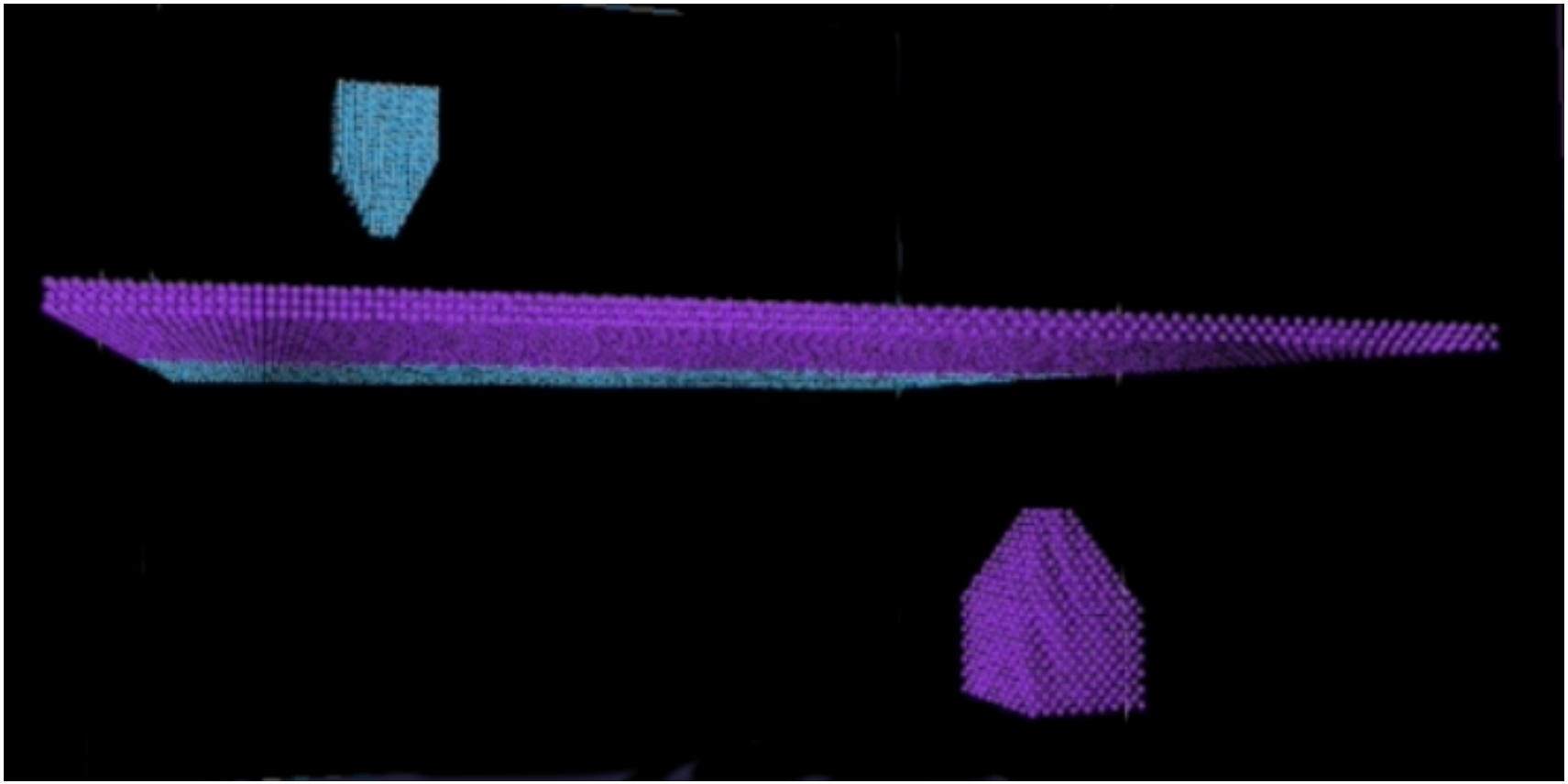

Figure : Overview of the interactive simulation (OpenGL + OpenCL app) with around 2 million particles

Summary

OpenCL programming model

What is OpenCL ?

- a standard for parallel programming of heterogeneous systems;
- initially influenced by GPU execution models;
- but now available on different architectures, including CPUs.

OpenCL portability

- the performance portability is not always guaranteed;
- because there are different HW designs (GPUs, CPUs, etc).

Do you need to have different optimizations for different devices ?

OpenCL programming model

Key terms

- Device - GPU, CPU, etc.;
- Work-item - Thread;
- Work-group - Group of work-items;
- Memory spaces:
  - Private - Work-item memory;
  - Local - Memory shared by work-items in a work-group;
  - Global - Memory shared by all work-items;
  - Constant - Read-only global memory.

OpenCL Runtime

- Device creation;
- Buffer management;
- Kernel dispatch.

OpenCL programming model

ScalVec kernel example

- vector *vec* is located in global memory;
- one work-item per vector element is used.

```
__kernel void ScalVec(__global float *vec, float k)
{
    int index = get_global_id(0);

    vec[index] *= k;
}
```

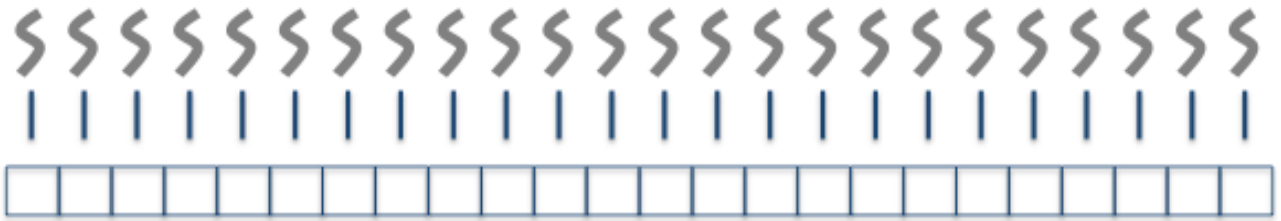

Figure : *ScalVec* kernel

NVIDIA GPU execution model

Streaming processor (SP)

- interleaved execution of sequential hardware threads;
- context switch is free (avoid stalling on memory load).

Streaming multiprocessor (SM)

- hosts groups of hardware threads;
- local memory sharing and synchronization.

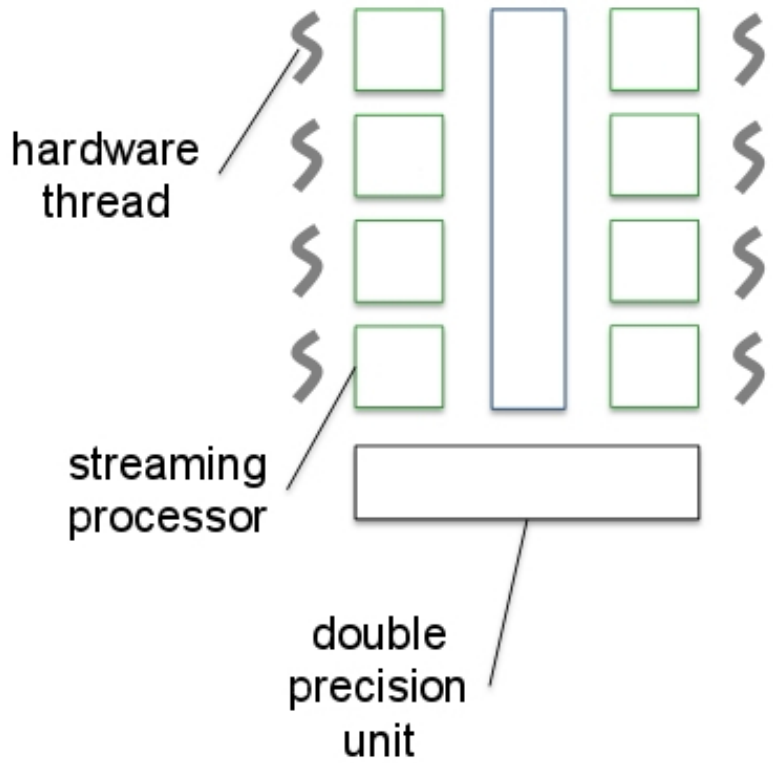

Figure : Cluster of SIMD units

Global memory is shared by all streaming multiprocessors

NVIDIA GPU execution model

Streaming multiprocessor

- several OpenCL work-groups can reside on the same SM;
- limited by hardware resources:
  - registers;
  - local memory;
  - max HW threads per SP.

Shared local memory

- much faster than global memory (shared by all SMs);
- only a few kBytes!

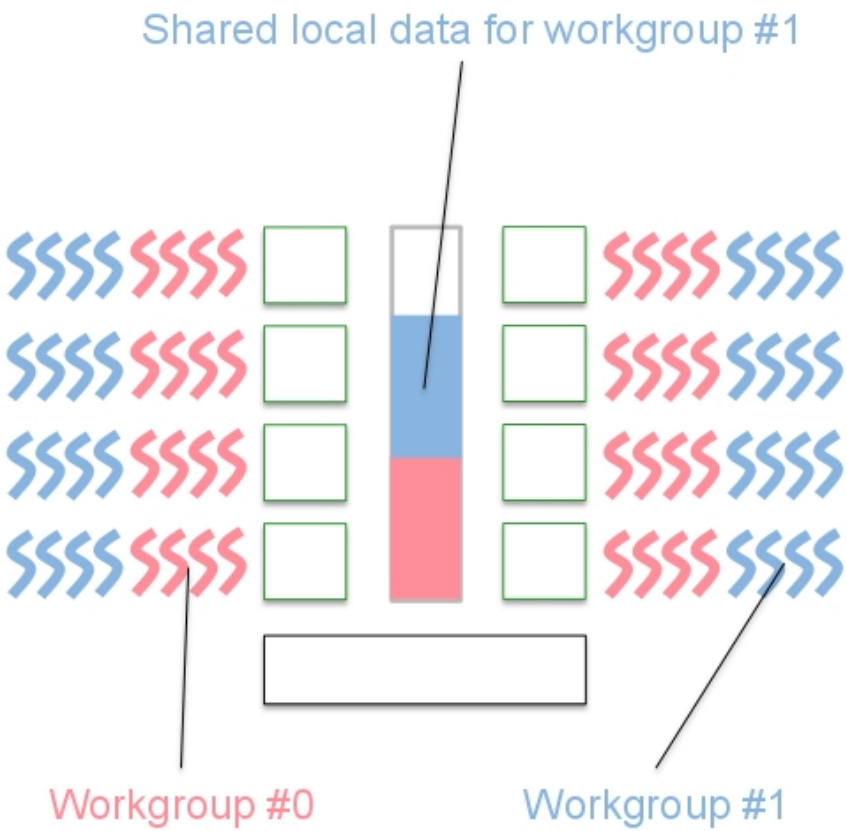

Figure : Streaming Multiprocessor

Intel Xeon Phi execution model

Xeon Phi & OpenCL

- 61 cores, 244 threads (4x threads interleaved);
- driver creates 240 SW threads which are pinned on each core:
  - threads scheduling in software (overhead).
- each work-group is executed sequentially by one thread.

Implicit vectorization

- kernels are implicitly vectorized along dimension 0;
- vector size of 16 elements.

```
__Kernel void foo(...)
  For (int i = 0; i < get_local_size(2); i++)
    For (int j = 0; j < get_local_size(1); j++)
      For (int k = 0; k < get_local_size(0); k += VECTOR_SIZE)
        Vectorized_Kernel_Body;
```

OpenCL best practices

NVIDIA GPU

- use tiling in local shared memory (much faster);
- memory accesses must be coalesced whenever possible;
- avoid different execution paths inside the same WG.

Intel Xeon Phi

- do not use local memory and avoid barriers:
  - no physical scratchpad local memory;
  - no special HW support, so barriers are emulated by OpenCL.
- code divergence may prevent successful vectorization;
- limit the number of kernels (software scheduling overhead).

Summary

Multi accelerators strategy

Initial version

- single accelerator version for NVIDIA GPUs;
  - developed by Raymond Namyst.

Objectives

- use multiple accelerators on a single node;
- distribute the work among accelerators;
- transfer particles between accelerators whenever it's needed:
  - to maintain physical properties (cf. cut-off distance).
- overlap memory accesses and optimize OpenCL code.

Distribute the work

How to split the 3D space ?

- spatial decomposition at the initialization;
- global domain splitted in Z plans of size r_c (cut-off distance).

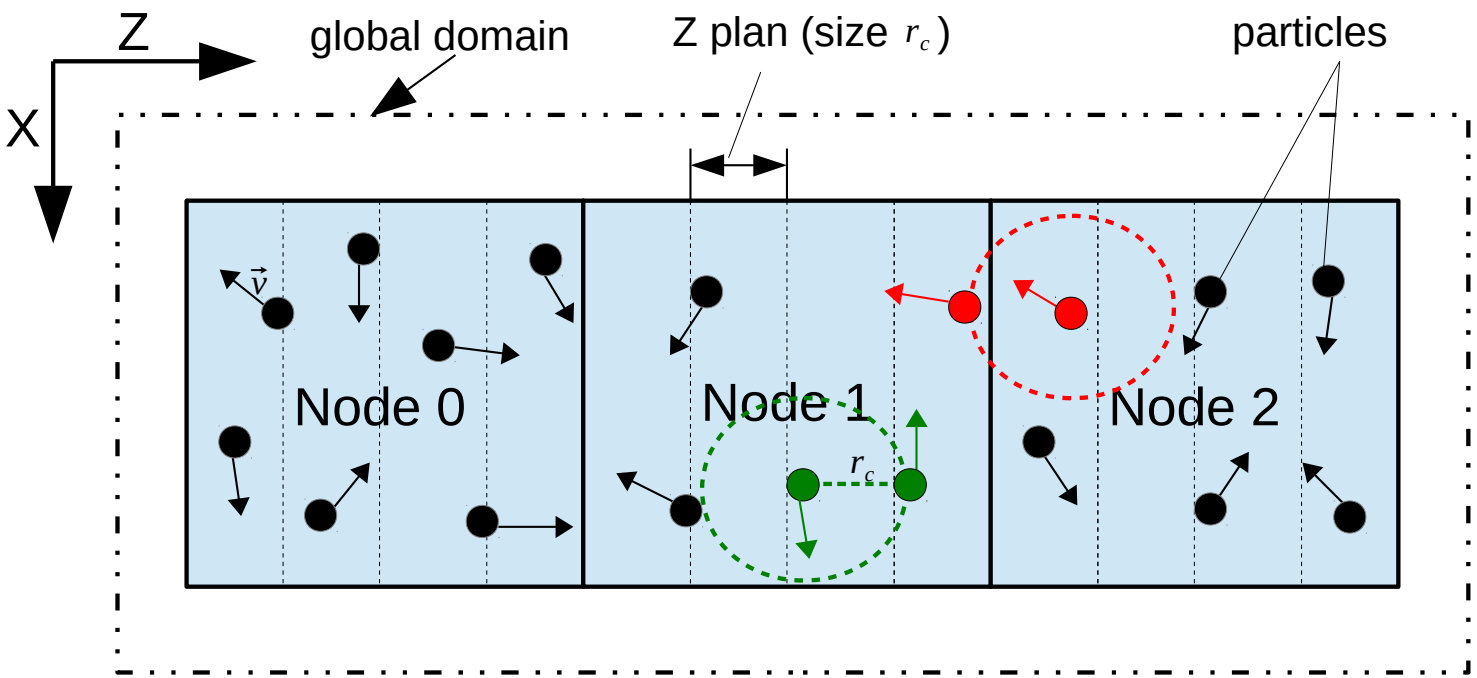

Figure : 2D overview of the spatial decomposition with 3 sub-domains

Transfer of particles

Borders management

- duplicate borders to maintain physical properties;
- a border is a Z plan with "ghost particles";
- "ghost particles" belong to a close sub-domain.

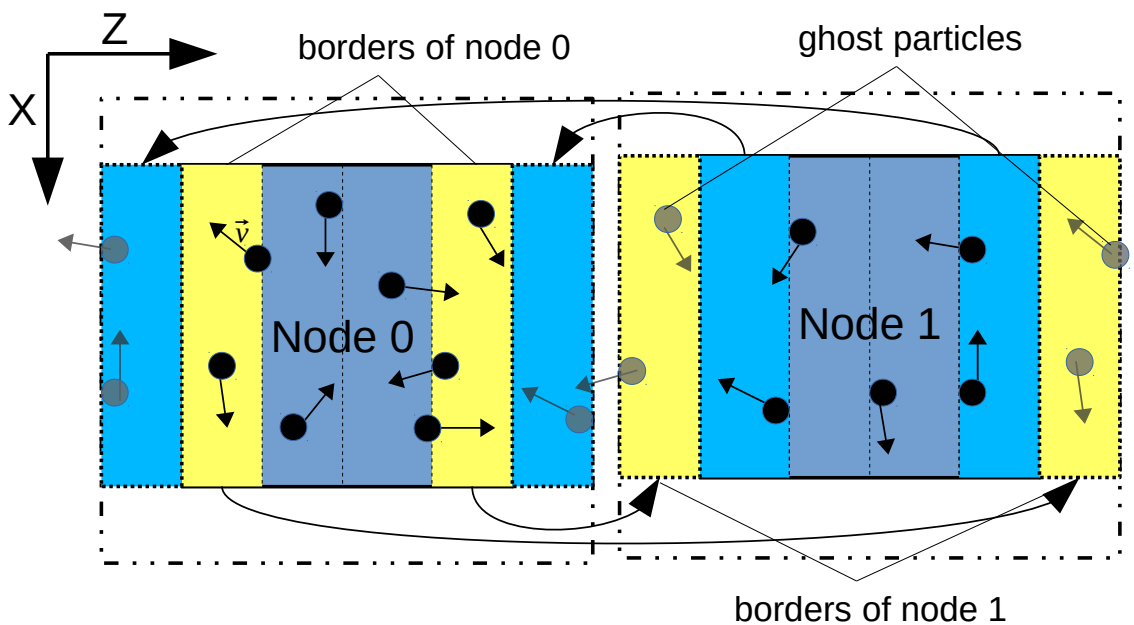

Figure : Exploded view of borders duplication with "ghosts particles"

Transfer of particles

Particles out-of-domain

- particles move during the simulation;
- a particle can move from a sub-domain to another one;
- need to transfer these particles after each iteration.

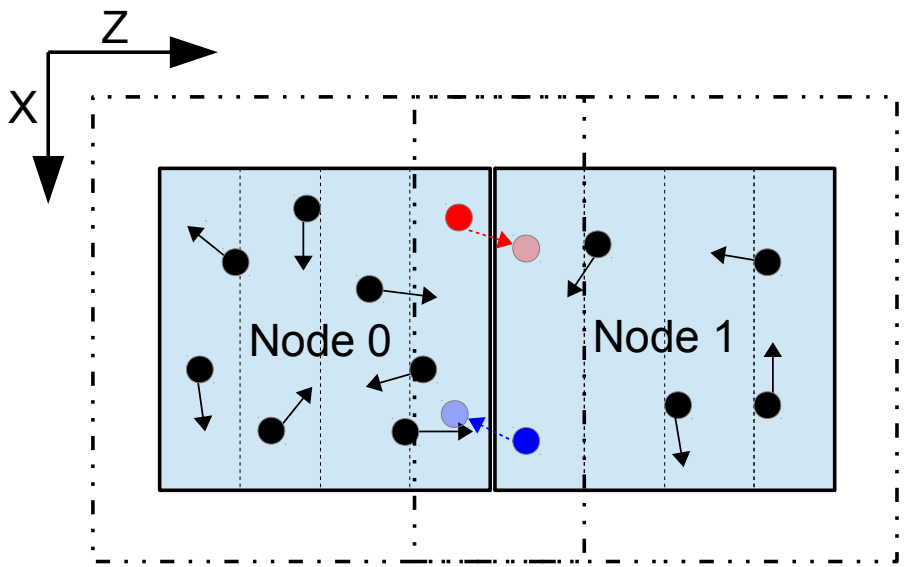

Figure : At the next step, the red particle will belong to the node 1, and the blue particle will belong to the node 0

Overlap memory accesses

Overlap memory accesses with HW computation

- parallel decomposition of the problem:
  - left and right borders are processed before the center;
  - allows to transfer borders while the center is processing.

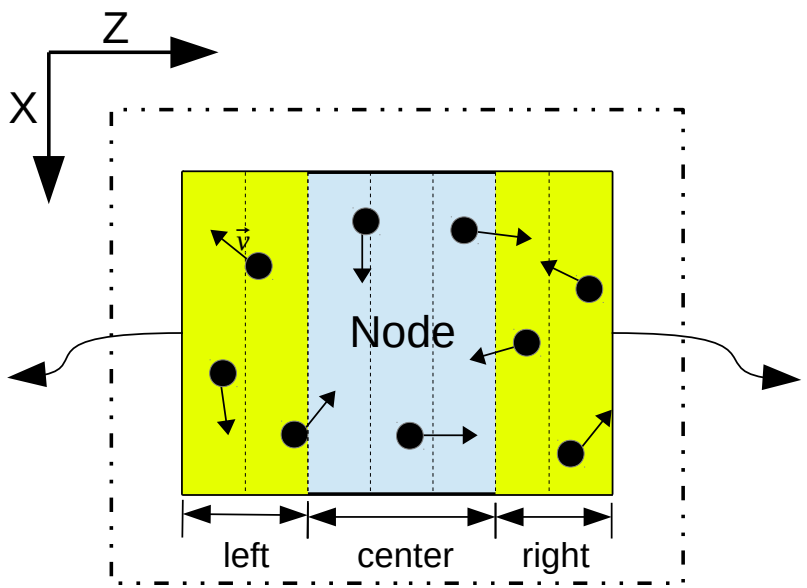

Figure : Parallel decomposition : left and right borders are processed before the center to allow to transfer borders

Parallelization strategy

Important points

- the most costly kernel;
- one thread per particle;
- 27 cells to compute forces with neighbors:
  - particles sorted at each iteration;
  - coalesced accesses along X axis.
- two implementations (GPU & CPU/MIC):
  - for performance & code readability.

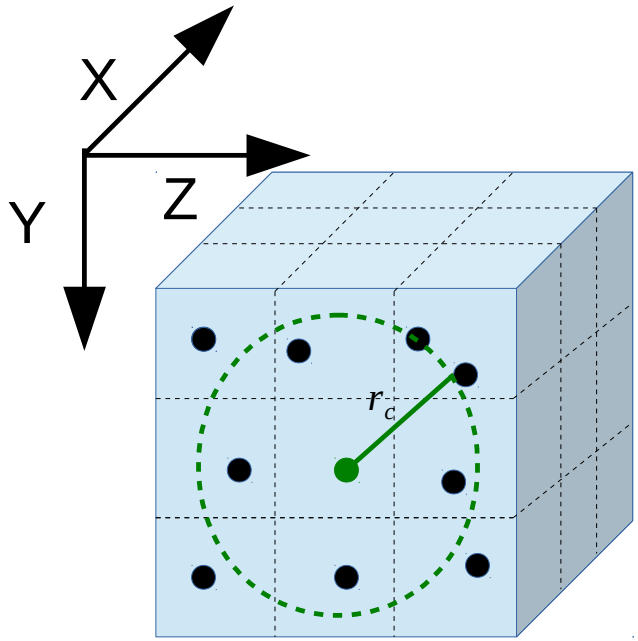

Figure : Computation of forces with neighbors (27 cells)

Summary

Limitations

- global domain needs to be homogeneous (static distribution);
- the slowest compute node slows down all others.

Discussion : load balancing

- idea: use a supervised learning based on execution times;
- profile performance of compute nodes;
- transfer Z plans between accelerators.

Summary

Single accelerator

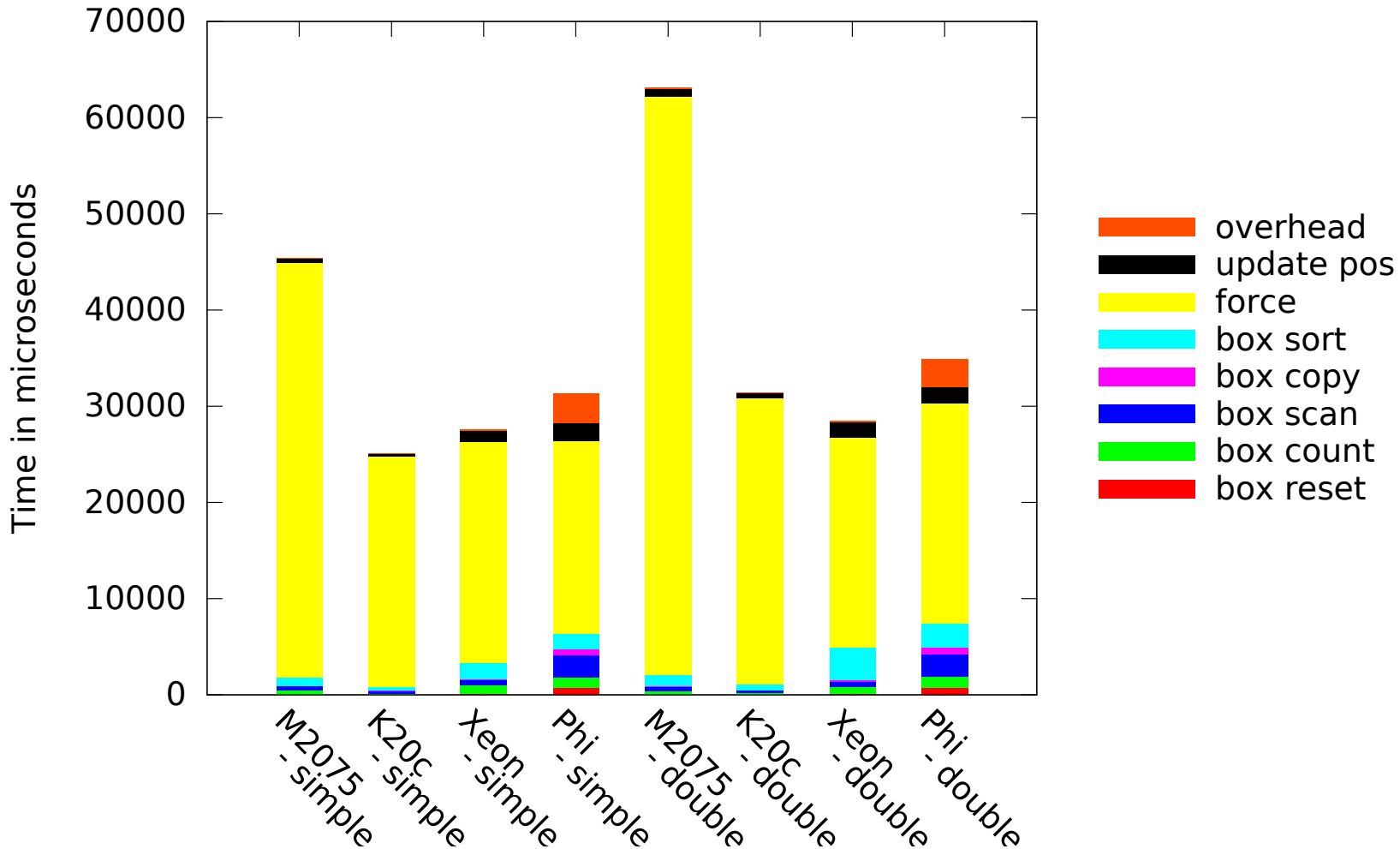

Figure : Time in microseconds for one iteration with one million particles in simple and double precision

Multi accelerators

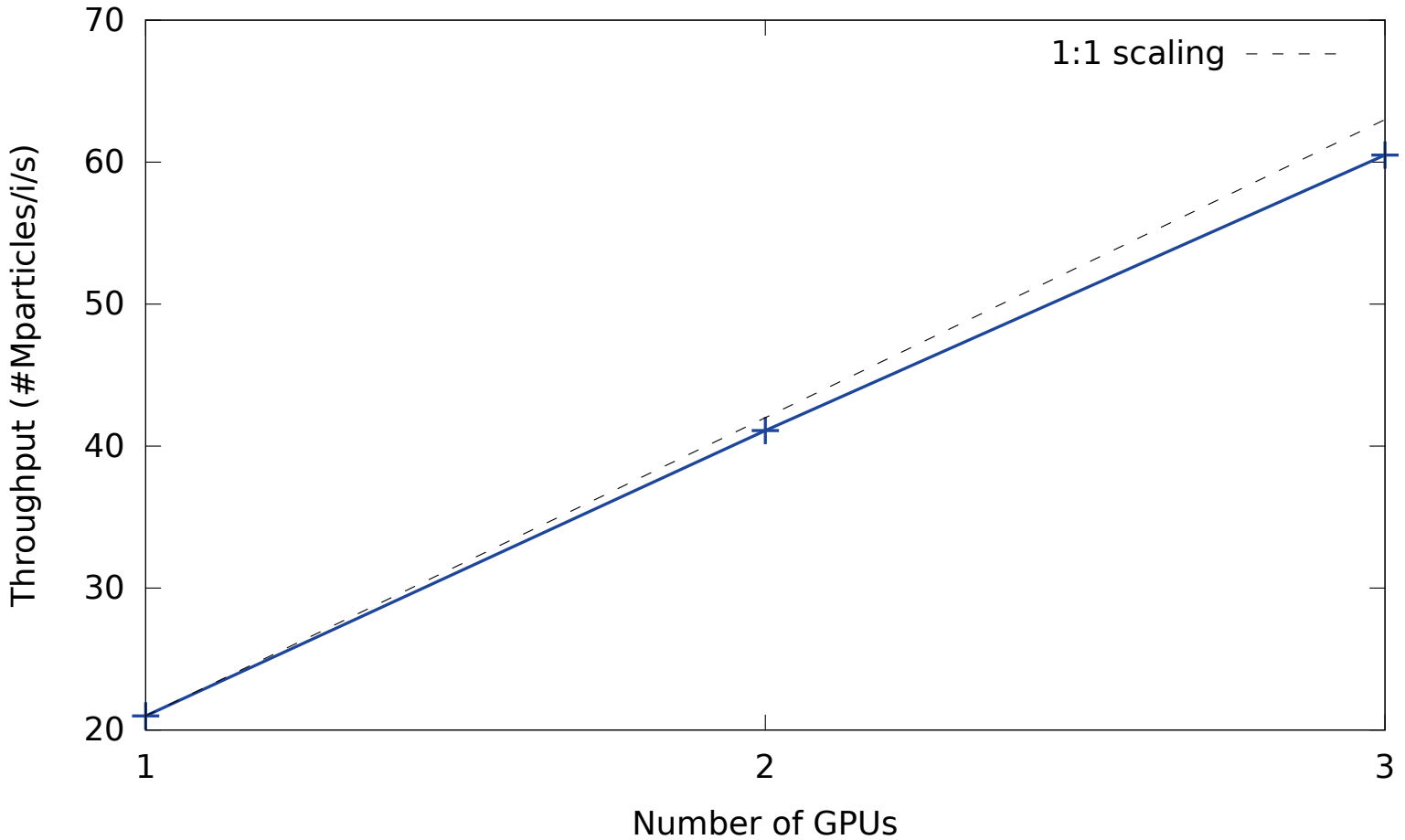

Figure : Throughput according to the number of GPUs (3xTesla M2075), in simple precision with around one million particles on each GPU

Summary

Conclusion

Current status

- more than 90M particles on accelerators with 5GB RAM;
- single precision performance results:
  - 61 Mparticles/i/s with 3xNVIDIA Tesla M2075 (gain: 2.9).
- works quite well with NVIDIA GPUs and Intel Xeon Phi.

Much potential (and ideas) for improvement

- load balancing between accelerators;
- some optimizations are still applicable on Xeon Phi;
- OpenCL kernels differ from one architecture to another:
  - OpenCL 2.0 could be a good start!

Questions & Discussions

Questions & Discussions