HPC strategies in Smilei

Julien Dérouillat

SMILEI training workshop
November 6-7, 2017
Maison de la Simulation

The HPC scale

HPC Concept	In Smilei
1. Distributed computing	1. MPI
2. Shared memory	2. OpenMP
3. Cache issues	3. Data sorting
4. Vectorization	4. SIMD / AVX
5. IO parallel	5. Parallel HDF5

Distributed computing : basics

- Based on a regular decomposition

  → even if many domains per processus …

- Using **MPI**
  - Mainly point-to-point comunications

    → Communications with neighbouring domains / time-step

    → particles displacement, explicit Maxwell solvers
  - Few collective communications

    → diagnostics, Poisson solver

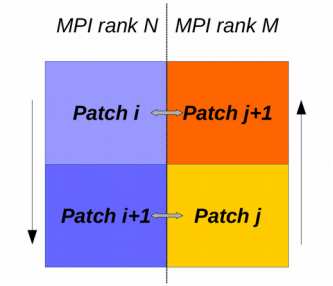
- Efficient scaling :
  - Occigen (*Haswell*) : 1000 cores → 50K cores
  - Turing (*Blue Gene Q*) : 8000 tasks → 64K tasks

Dynamic load balancing

- Balanced number of particles on **MPI process**
  - Many small domains called « patch » per process
- Dynamic : Exchange of « patch » between process
  - Particles & cells (fields & boundary conditions)
  - Constraint to force locality : minimize cost, @ high frequency
- Communications/Synchronizations @ patch level
  - Many comms/syncs per process for a given physical quantity

    → Asynchronous communications based on tags

    → Dedicated buffers per patch
  - MPI or memcpy



- Along a space filling curve : favour memcpy

Dynamic load balancing

```
Main.number_of_patches=[ 2^n, 2^m, 2^p ]
```

- Balanced number of particles on **MPI process**
  - Many small domains called « patch » per process
- Dynamic : Exchange of « patch » between process

```
LoadBalancing.every
```

  - Constraint to force locality : minimize cost, @ high frequency
- Communications/Synchronizations @ patch level
  - Many comms/syncs per process for a given physical quantity

```
LoadBalancing.initial_balance
LoadBalancing.cell_load
LoadBalancing.frozen_particle_load
```

  - MPI or memcpy
- Along a space filling curve : favour memcpy

Shared memory

- **OpenMP** : medium grain implementation
  - Computation shared between **OpenMP threads** @ patch level

```
#pragma omp for
for (i=0 ; i<number_of_patches ; i++)
   patch[i]→push_particles();
```

- OpenMP parallelism reachable
  - For all local PIC operators
  - Communications/Synchronizations

```
#pragma omp for / number_of_patches
   patch[i]→init_MPI_communications();
#pragma omp for / number_of_patches
   patch[i]→local_copy( neighbors[i] );
#pragma omp for / number_of_patches
   patch[i]→finalize_MPI_communications();
```

→ `MPI_THREAD_MULTIPLE` required

OpenMP load balancing

- Particles : Dynamic scheduling

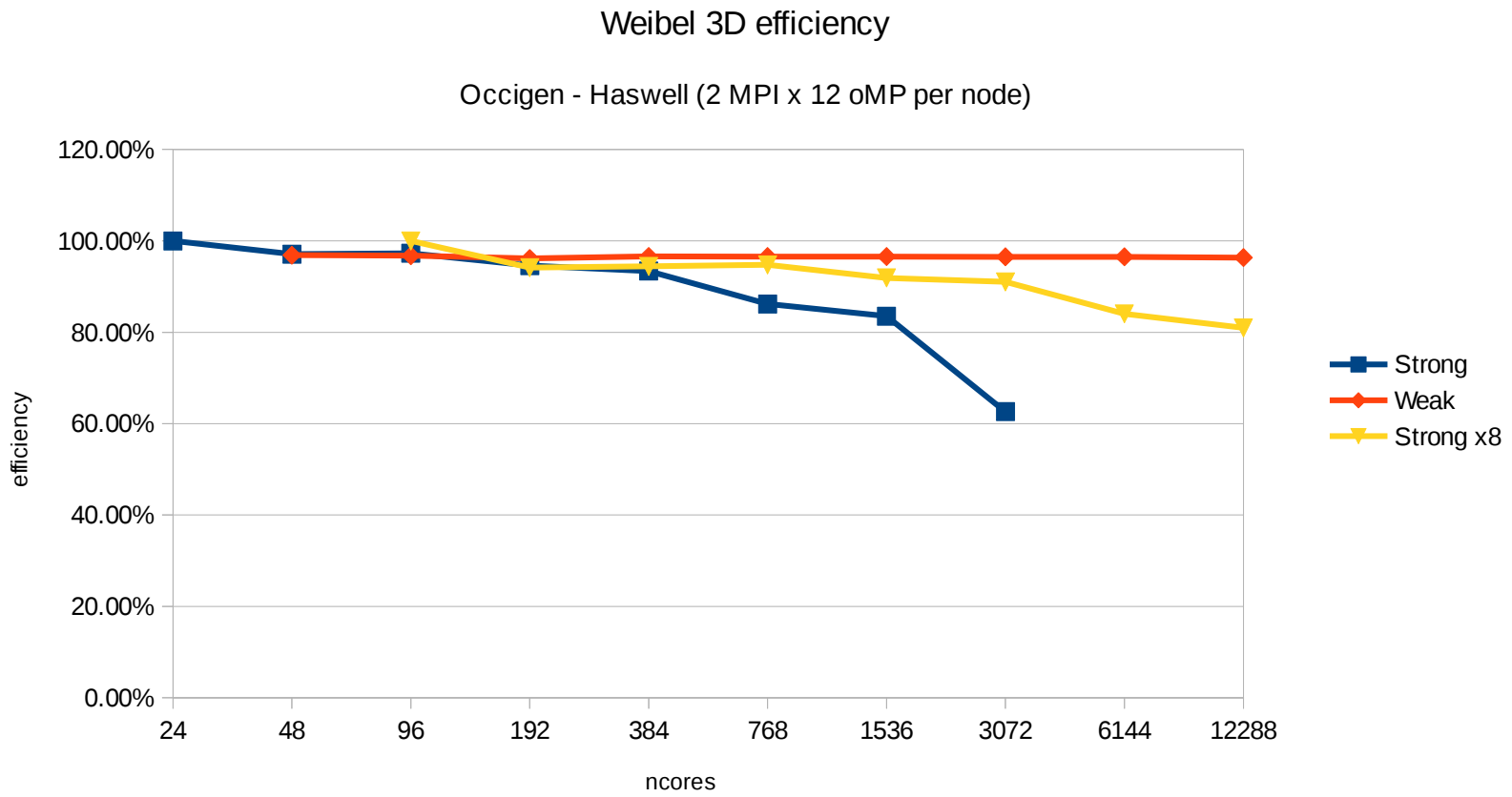
```
export OMP_SCHEDULE=dynamic

#pragma omp for schedule(runtime)
for (i=0 ; i<number_of_patches ; i++)
   patch[i]→push_particles();
```

- Cells : Static scheduling

```
#pragma omp for schedule(static)
for (i=0 ; i<number_of_patches ; i++)
   patch[i]→solve_Maxwell();
```

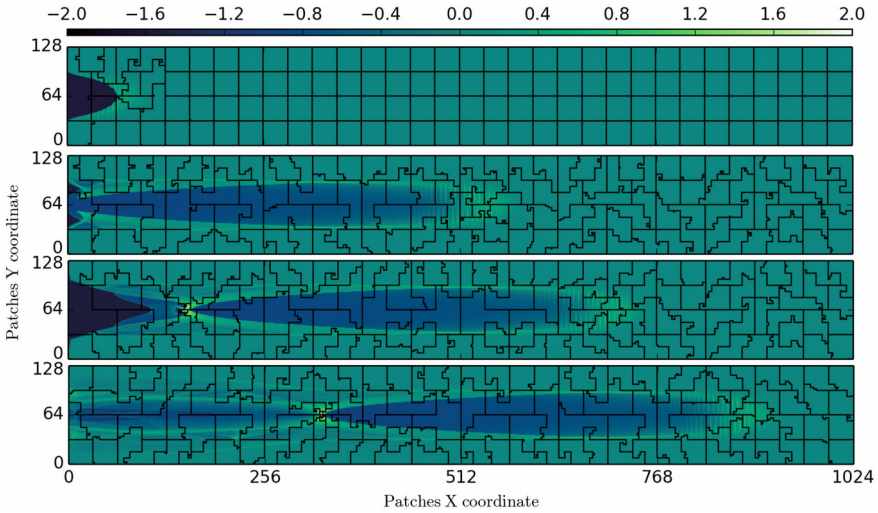
MPI x OpenMP Scaling

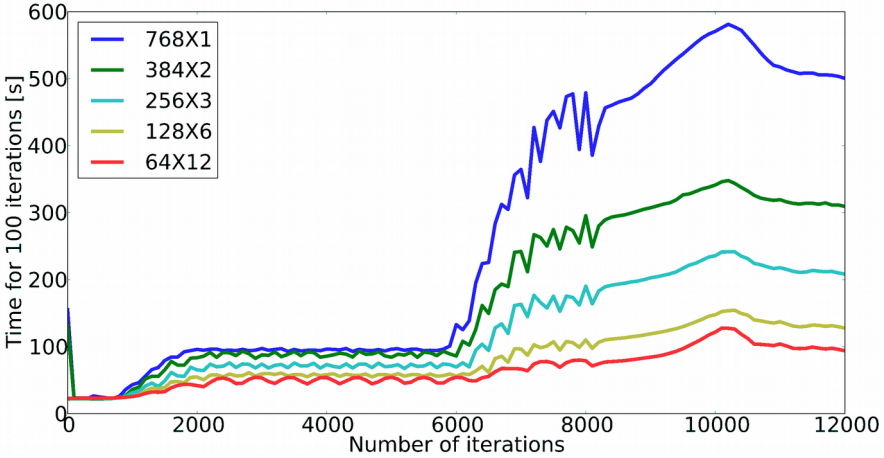


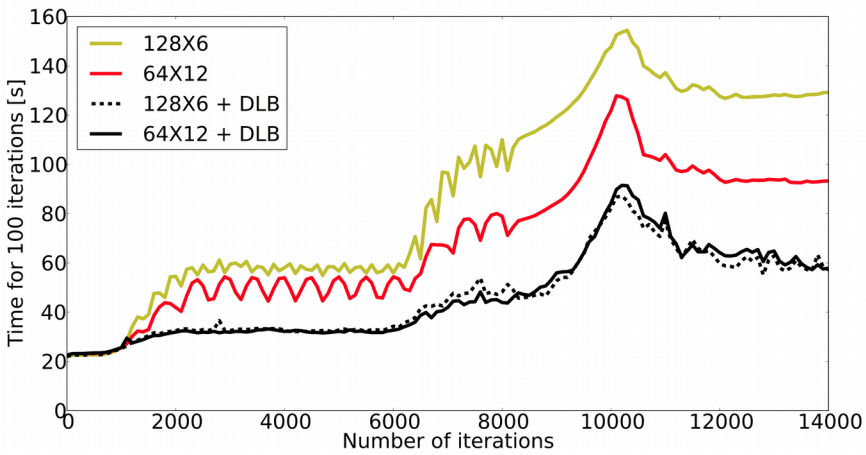
Strong scaling : Fixed global size
- Strong : 2048 patches, number of cells = 128^3, number of particles : 226 millions
- Strong x8 : 16384 patches, number of cells = 256^3, number of particles : 1.8 billions

Weak scaling : Fixed per process size
- 1024 patches per MPI process, 8^3 cells per patch, 55 thousands particles per patch

Electrons acceleration : Load balancing





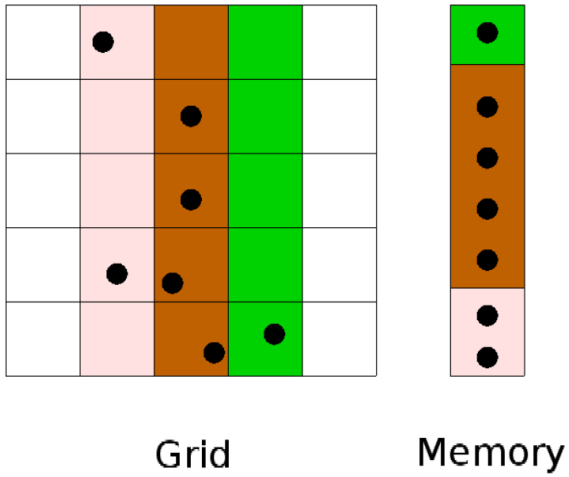


Data locality

Concerns particles-grid interactions

- Interpolation / Projection (*field gathering / current deposition)*
- Particles sorting
- Clustering :

→ Particles of a given cluster access local region of memory



Location	Access cost
Cache L1	~4 cycles
Cache L2	~10 cycles
Cache L3	~40-65 cycles
Memory	60ns (150-200 cycles)

`Main.clrw`

Vectorization for new architecture

Not yet released

SIMD : Single Instruction Multiple Data

- Curie/Occigen : **AVX**, register width = 256 Bit

  → 4 double precision operands

- Frioul, *Irène* : **AVX-512**, register width = 512 Bit

  → 8 double precision operands

  – *Irène : May 2018*

    → *1656 x 2 x 24 cores Skylake*

    → *684 x 68 cores KNL*

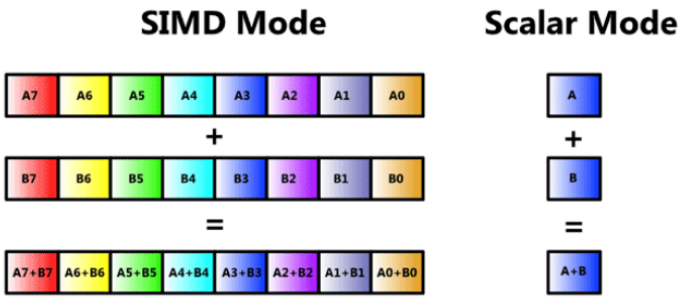


Figure 3. SIMD versus scalar operations

- Contiguous memory layout

  – Particle**s** only operator : SoA

    - C++ class containing many `std::vector`

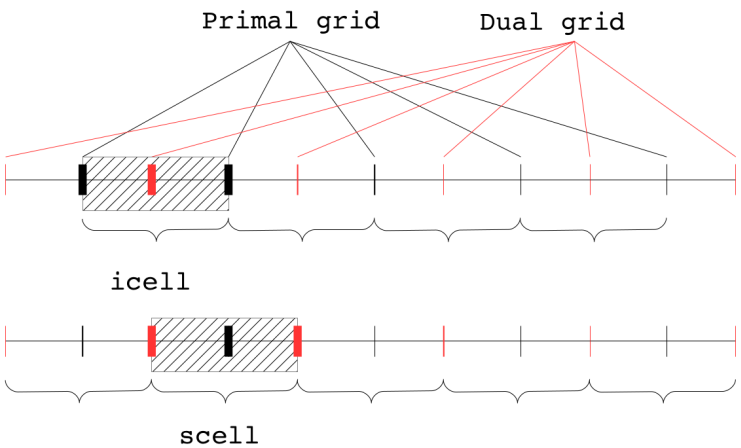
  – Cells only operator : C arrays

SIMD : Interpolation

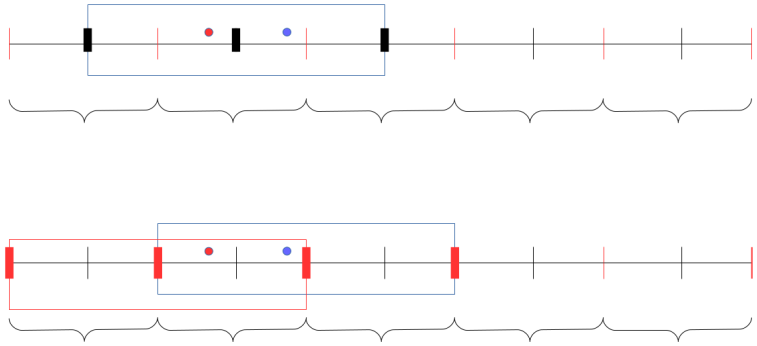
Not yet released

- Thanks to a per cell hierarchical sorting
- 2nd order Interpolation :
  - All particles of the cell have the same primal index
  - Along a primal direction : 3 points
  - Along a dual direction : 4 points with 1 null for each particle

Null points defined with(out) gap between primalDual indexes



Sorting principle



Interpolation : primal and dual grids (worst case)

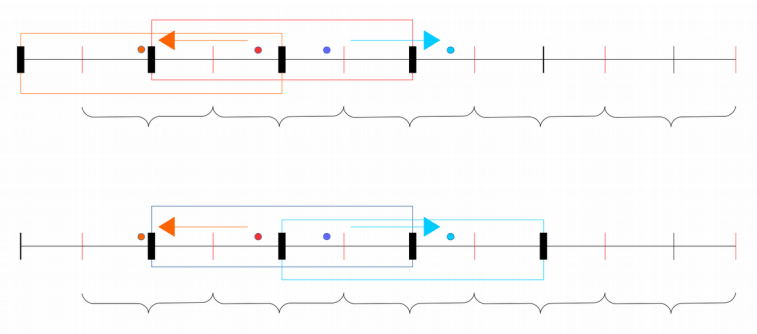
SIMD : Projection

Not yet released

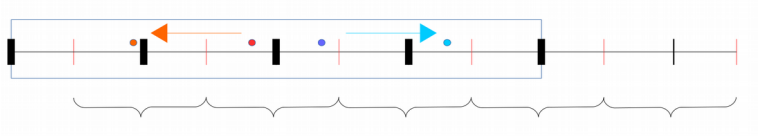
- 2nd order Projection using Esirkepov
- Same problem as scalar condition
  - Primal grid only
  - But particles can be pushed to +/- dx

→ Up to 4 points among 5

- Light overhead to manage gap for new positions



Extremal cases :
- up : particle pushed @ - dx
- down : particle pushed @ + dx



Projection zone for a sorted cell :
→ Maximum 5 points

SIMD : Projection algorithm

Not yet released

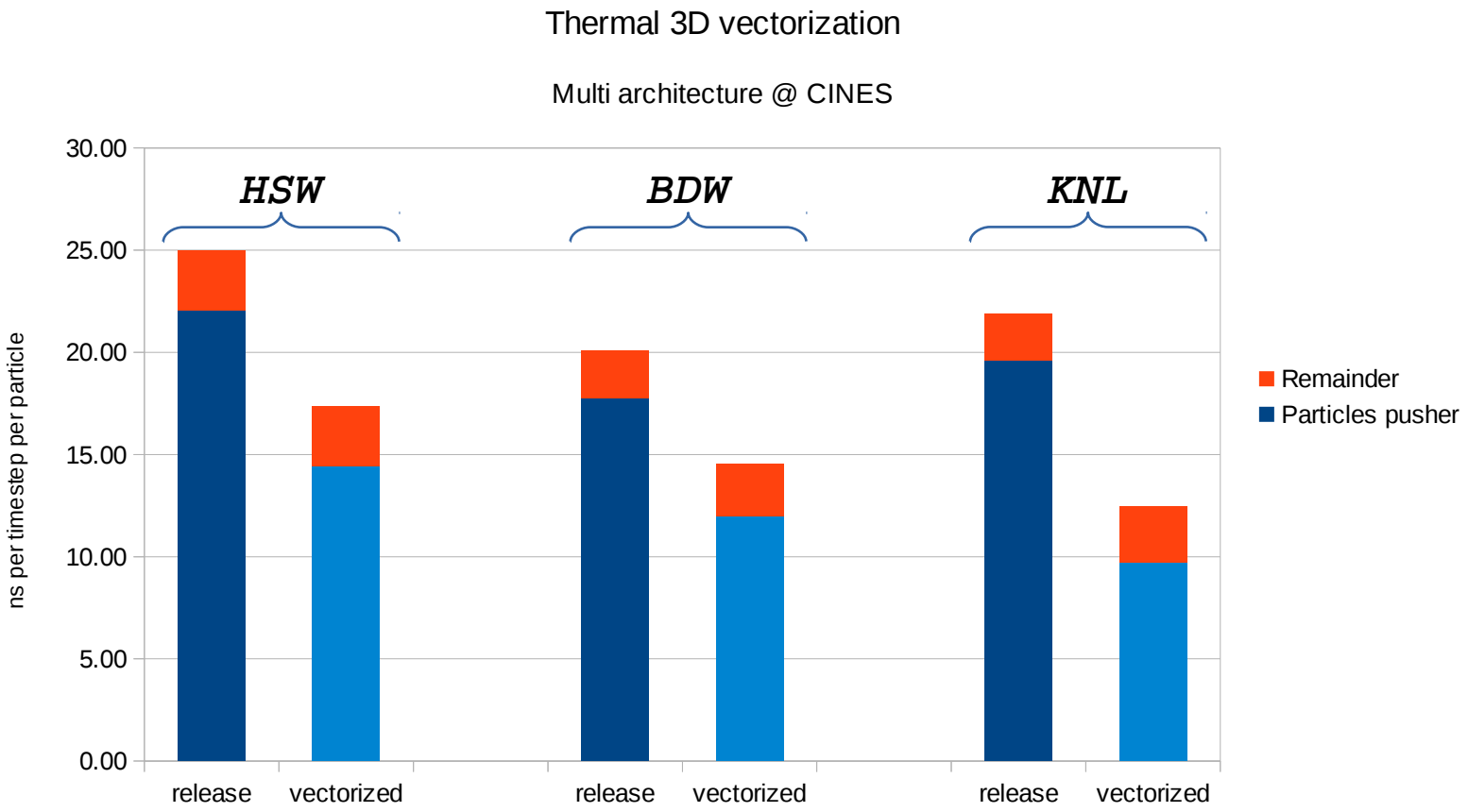
Reduction : avoid write conflicts on particles

```
Loop vectors of particles( scell )
    #pragma omp simd
    Loop particles / vector
      - Compute coefficients / positions pre-push
      - Compute coefficients / positions post-push
        (gap)
      - Store contributions in buffers
            particles major storage

    #pragma omp simd
    Loop cells / zone( projection )
        #pragma unroll
        Loop particles / vector
            Backport stored contributions in cells
```

SIMD : Results

Not yet released

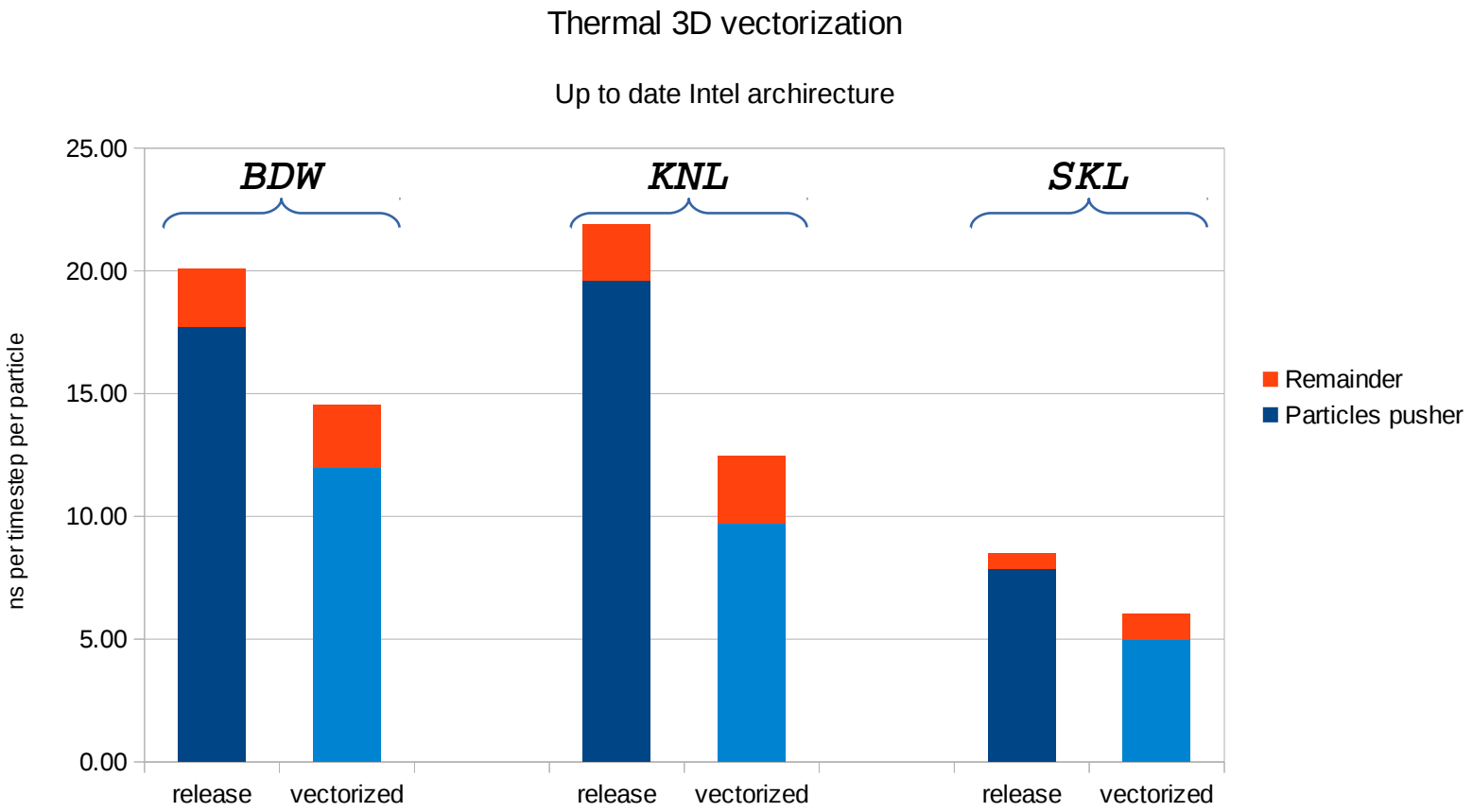


16^3 patchs, full MPI
Grid size : 64^3
Number of particles : 33 millions

HSW : 2 x 12 cores @ 2.6 Ghz
BDW : 2 x 14 cores @ 2.6 Ghz
KNL : 68 cores x 2 HT @ 1.4 Ghz, cache mode

SIMD : TGCC projection

Not yet released

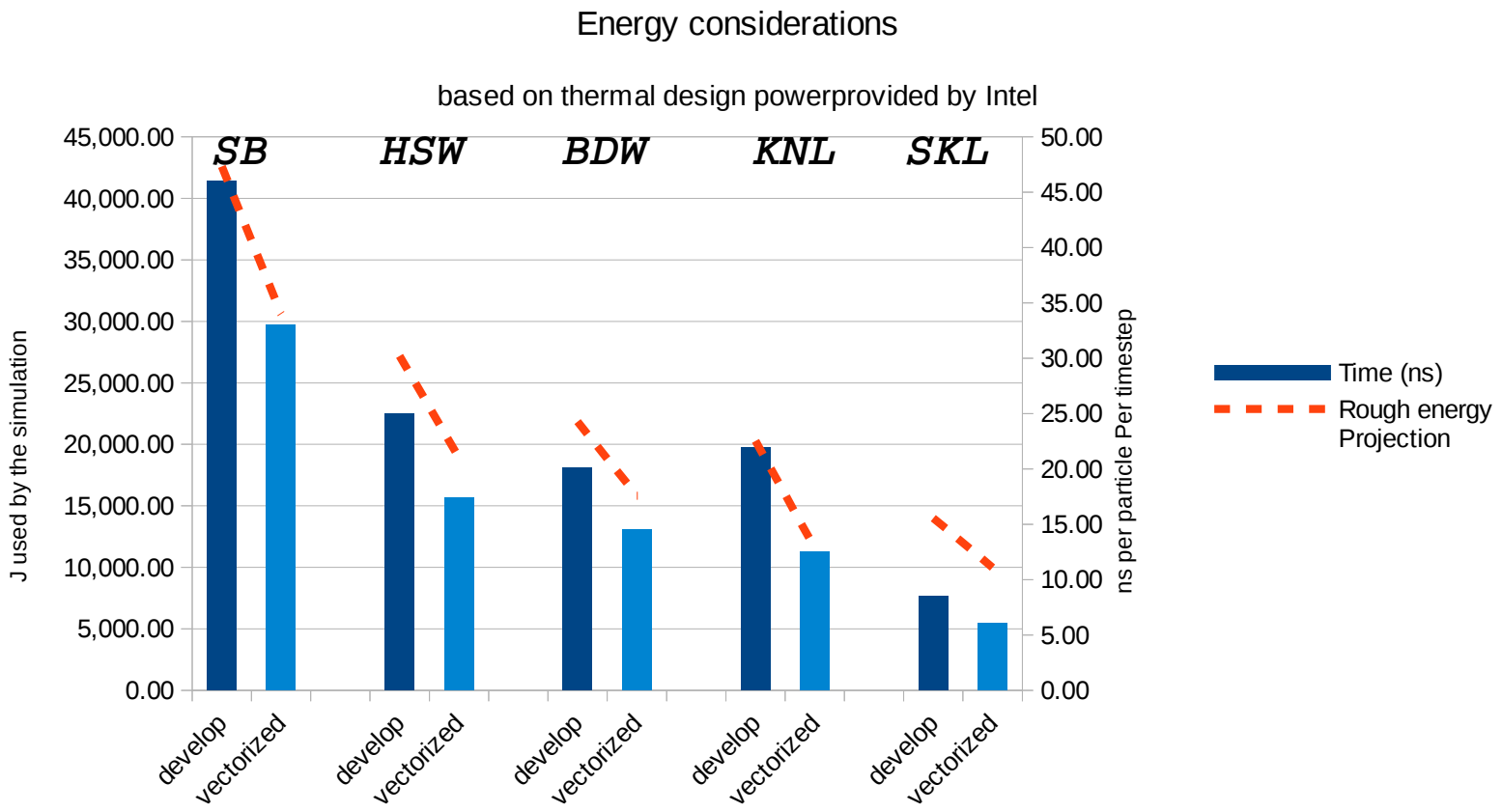


16^3 patchs, full MPI
Grid size : 64^3
Number of particles : 33 millions

BDW : 2 x 14 cores @ 2.6 Ghz
KNL : 68 cores x 2 HT @ 1.4 Ghz, cache mode
*SKL : 2 x 24 cores @ 2.7 Ghz (rel : **AVX-2**, vecto : **AVX-512**)*

Energy considerations

Not yet released



SB : 2 x 8 cores @ 2.6 Ghz, 230 W
HSW : 2 x 12 cores @ 2.6 Ghz, 270 W
BDW : 2 x 14 cores @ 2.6 Ghz, 270 W
KNL : 68 cores x 2 HT @ 1.4 Ghz, cache mode, 230 W
SKL : 2 x 24 cores @ 2.7 Ghz (dev : AVX-2, vecto : AVX-512), 410 W

Distributed data

- Generate consistent output files
- 2 categories :
  - Reductions diagnostics :

    `Diagnostic[ Scalar, Screen ], ParticleBinning`
  - Shared diagnostics :

    `Diagnostic[ Fields, Probes, Track ]`
- In all case :
  - Hierarchical computation : local, then global :
    - **`MPI_Reduce`**
    - **`H5Dwrite`** using collective I/O access **`H5FD_MPIO_COLLECTIVE`**
- On Parallel File System : LUSTRE, GPFS
  - File striping, **`MPIIO_HINTS`**

Perspectives

- Integration of vectorization in a release in 2017
- Implementation of a grid uncoupled from particles
  - Spectral solvers, see H. Kallala
  - Grid based diagnostics
  - Static load balancing efficiency
- Test of the unified parallel framework : **MPC**
  - Improve interleaved OpenMP and MPI synchronizations

    http://mpc.hpcframework.paratools.com/
- Dedicated resources for diagnostics and IO
  - IO and parallel post processing servers
  - **XIOS** developed and used by the climate community (LSCE)