Efficient Strict-Binning Particle-in-Cell (PIC) Algorithm for Multi-Core SIMD Processors

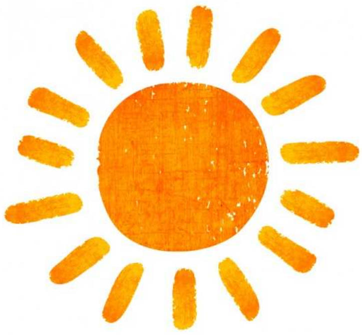
Yann Barsamian[1,2], Arthur Charguéraud[2,1], Sever Hirstoaga[3,1], Michel Mehrenberger[1,3]

1. Université de Strasbourg
2. ICube, CNRS, INRIA Nancy
3. IRMA, CNRS, INRIA Nancy

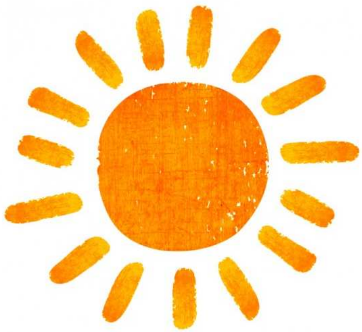
Euro-Par 2018, Torino (Italy)
August 2018

General Context: Controlled Thermonuclear Fusion

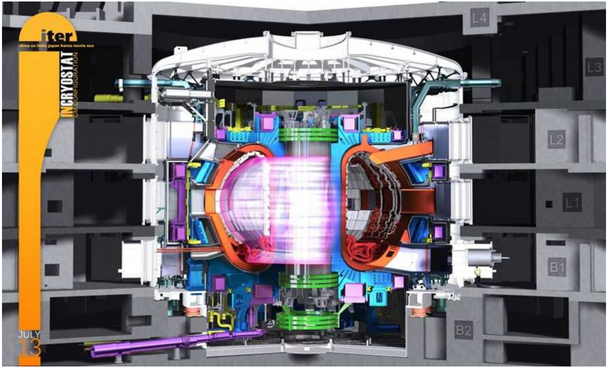
Step 1.



Step 2.



General Context: Controlled Thermonuclear Fusion



Step 3. ITER[1] tokamak[2]

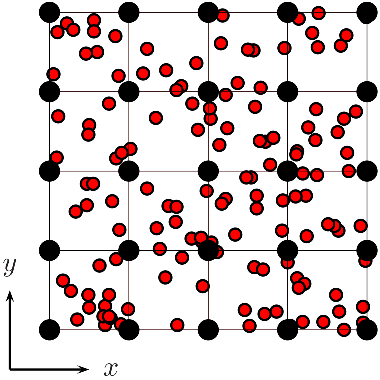
(also applicable in other contexts, *e.g.*, astrophysics, where we have to model different particles / planets / ... that interact)

[1] "The way" (in Latin) to produce energy (Cadarache, France)

[2] Токамак: тороидальная камера с магнитными катушками (toroidal chamber with magnetic coils)

$$\begin{cases} \dfrac{\partial f}{\partial t} + \overrightarrow{v} \cdot \nabla_{\overrightarrow{x}} f - \overrightarrow{E} \cdot \nabla_{\overrightarrow{v}} f = 0 & \text{Vlasov} \\ \nabla_{\overrightarrow{x}} \overrightarrow{E} = \rho = 1 - \displaystyle\int f \, d\overrightarrow{v} & \text{Poisson} \end{cases}$$

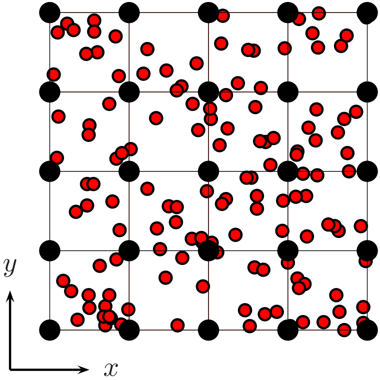
- Distribution function f: N numerical particles (red)
- Electric field $\overrightarrow{E}$ and charge density ρ: 3d grids (black)



Kinetic Modeling with Particle-in-Cell (PIC) Methods

$$\begin{cases} \dfrac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f - \vec{E} \cdot \nabla_{\vec{v}} f = 0 & \text{Vlasov} \\ \nabla_{\vec{x}} \vec{E} = \rho = 1 - \displaystyle\int f \, d\vec{v} & \text{Poisson} \end{cases}$$

- Distribution function f: N numerical particles (red)
- Electric field $\vec{E}$ and charge density ρ: 3d grids (black)

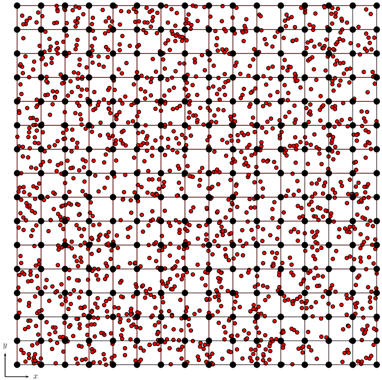


- Physical effects on small scale (+ large scale)
- Noise (numerical errors when N is small)
- Frequent particle motion

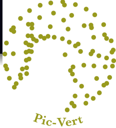
Kinetic Modeling with Particle-in-Cell (PIC) Methods

$$\begin{cases} \dfrac{\partial f}{\partial t} + \overrightarrow{v} \cdot \nabla_{\overrightarrow{x}} f - \overrightarrow{E} \cdot \nabla_{\overrightarrow{v}} f = 0 & \text{Vlasov} \\ \nabla_{\overrightarrow{x}} \overrightarrow{E} = \rho = 1 - \displaystyle\int f \; d\overrightarrow{v} & \text{Poisson} \end{cases}$$

- Distribution function f: N numerical particles (red)
- Electric field $\overrightarrow{E}$ and charge density ρ: 3d grids (black)

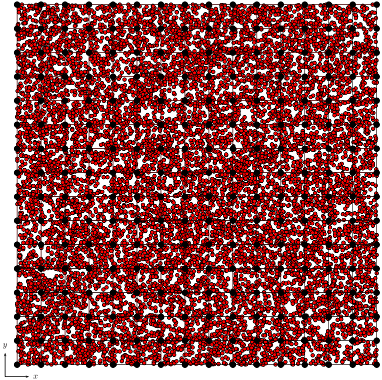


- Physical effects on small scale (+ large scale)
  $\Rightarrow$ increase $ncx \times ncy \times ncz$
  (1 000 $\times$ 1 000 $\times$ 1 000)
- Noise (numerical errors when N is small)
- Frequent particle motion

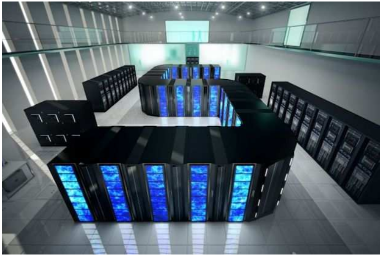


$$\begin{cases} \dfrac{\partial f}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} f - \vec{E} \cdot \nabla_{\vec{v}} f = 0 & \text{Vlasov} \\ \nabla_{\vec{x}} \vec{E} = \rho = 1 - \displaystyle\int f \, d\vec{v} & \text{Poisson} \end{cases}$$

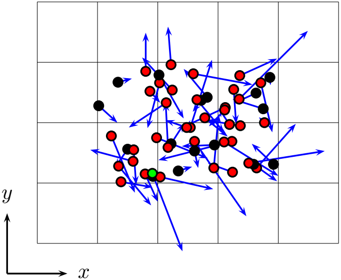
- Distribution function f: N numerical particles (red)
- Electric field $\vec{E}$ and charge density ρ: 3d grids (black)



- Physical effects on small scale (+ large scale) $\Rightarrow$ increase $ncx \times ncy \times ncz$ $(1\ 000 \times 1\ 000 \times 1\ 000)$
- Noise (numerical errors when N is small) $\Rightarrow$ increase $\frac{N}{ncx \times ncy \times ncz}$ (10 000 to 1 000 000)
- Frequent particle motion



- Three levels of parallelism :
  - network (`MPI`, inter-node),
  - socket (`OpenMP`, intra-node),
  - instruction (SIMD),
- Maximization of the number of particles that can fit in memory,
- Maximization of the throughput of the simulation which is memory bound,
- Handling particles moving more than 2 cells per time step ("fast-moving particles"), without loss of performance,
- Comparison to other implementations.



Particle-in-Cell (PIC) Pseudo-Code

Initialization:

Initialize N particles — `icell, d{x,y,z}, v{x,y,z}` of size `[N]`
Compute ρ and E — `rho, E{x,y,z}` of size `[ncx][ncy][ncz]`

Algorithm:

Foreach time iteration **do**
    If (*condition*) **then**
        Sort the particles[3] — $\mathcal{O}(N)$ counting sort
    End If
    Set all cells of ρ to 0
    Foreach particle **do**
        Update the velocity — $v+=-E\Delta t$
        Update the position — $x+=v\Delta t$
        Accumulate the charge on the nearest ρ cells
    End Foreach
    Compute E from ρ — FFT Poisson solver
End Foreach

[3] Decyk, Karmesin, de Boer, & Liewer (1996)

Particle-in-Cell (PIC) Pseudo-Code

Initialization:

Initialize N particles — `icell, d{x,y,z}, v{x,y,z}` of size `[N]`

Compute ρ and E — `rho, E{x,y,z}` of size `[ncx][ncy][ncz]`

Algorithm: — Execution time breakdown

Foreach time iteration **do**

    If (*condition*) **then**

        Sort the particles[3] — 10%[4]

    End If

    Set all cells of ρ to 0

    Foreach particle **do**

        Update the velocity — 50%[4]

        Update the position — 25%[4]

        Accumulate the charge on the nearest ρ cells — 15%[4]

    End Foreach

    Compute E from ρ — $<$1%[4]

End Foreach

[3] Decyk, Karmesin, de Boer, & Liewer (1996)

[4] Any difference in system hardware or software design or configuration may affect actual performance (-:

To sort or not to sort?

	Sort	Upd. v	Upd. x	Deposit	Total
Do not sort	0.0	98.0	64.6	35.9	199.0
Sort every 100	3.6	78.3	64.4	25.6	177.0
Always sort	209.0	66.3	64.2	13.4	353.0

Execution time (in s). Test case: 200 000 000 particles, 128×128 grid, $\Delta t = 0.1$, 500 iterations. Architecture: Intel Broadwell, 18 cores, 76.8 GB/s.

Periodic sorting: better data locality, and shorter overall time: find the best frequency[5].

Sorting at each iteration[6]: enhancement of the data locality & vectorization of the update velocities loop, but too costly.

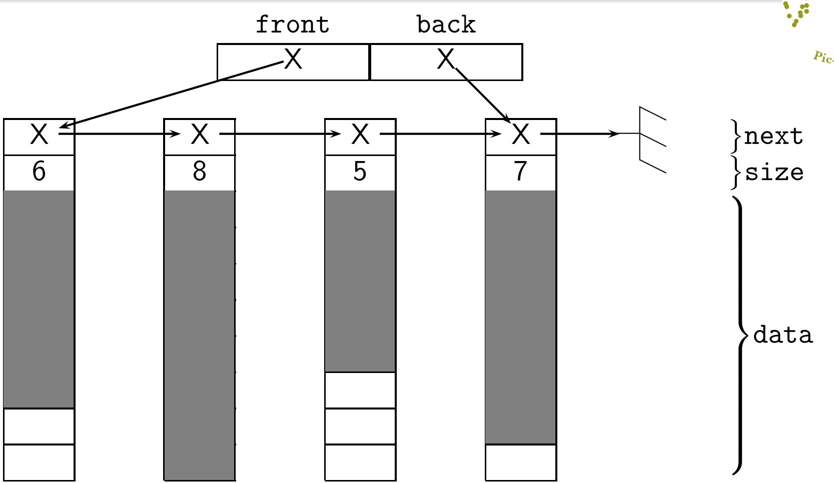
Efficient data structure to keep particles sorted[7]: avoid the sorting step.

[5] Marin, Jin, & Mellor-Crummey (2008)

[6] Lanti, Tran, Jocksch, Hariri, Brunner, Gheller, & Villard (2016)

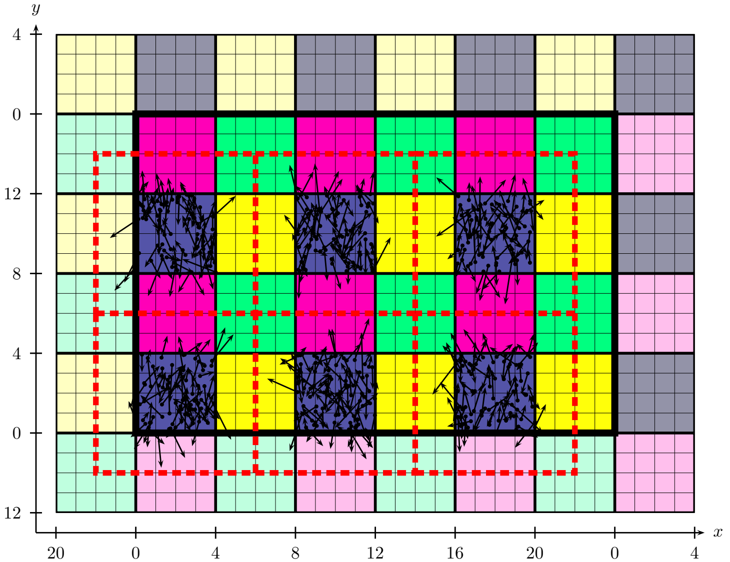
[7] Durand, Raffin, & Faure (2012); Nakashima, Summura, Kikura, & Miyake (2017); Barsamian, Charguéraud, & Ketterlin (2017)

Chunk Bags: Linked Lists of Fixed-Size Arrays




```
struct chunk { struct chunk* next; int size; // 0<=size<=K
               float  dx[K], dy[K], dz[K];
               double vx[K], vy[K], vz[K]; } chunk;
struct { chunk* front, back; } bag;
```

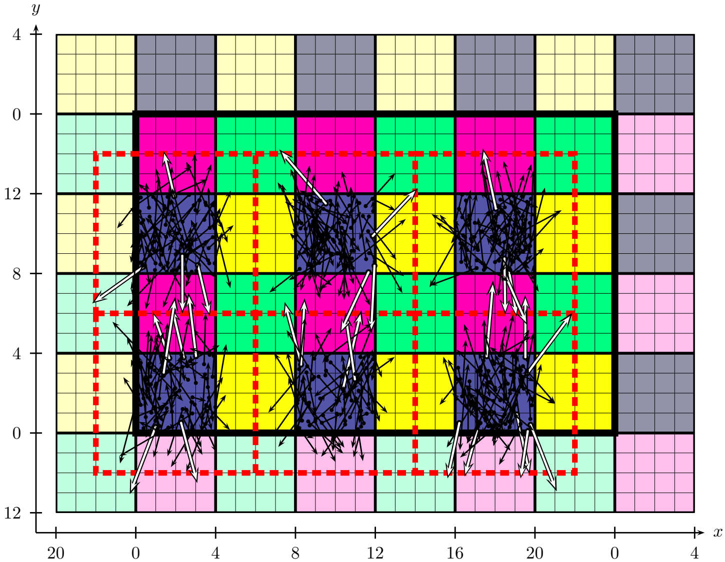
The Eight-Colors Algorithm[8]



8 phases to tame the number of data races when moving particles.

[8]Kong, Huang, Ren, & Decyk (2011)

The Eight-Colors Algorithm[8]

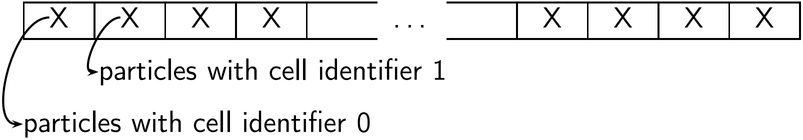


Particles moving more than half a tile away require special care.

[8] Kong, Huang, Ren, & Decyk (2011)

Chunk Bags: Particle Arrays

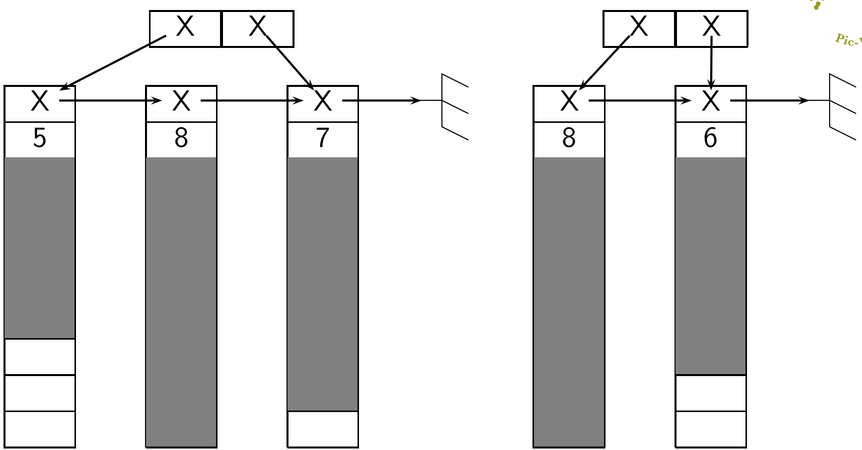
```
chunkbag particles[nbCells] // nbCells = ncx*ncy*ncz
```



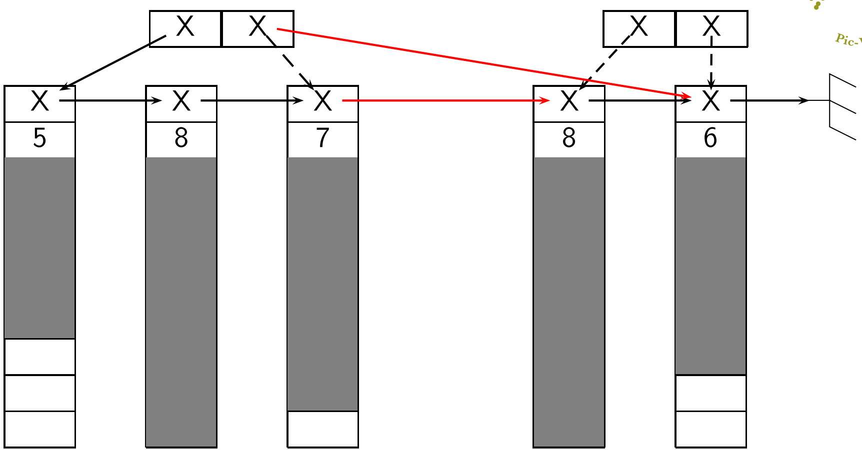
```
chunkbag particlesNextPrivate[nbCells],
         particlesNextShared[nbCells]
```

- `particlesNextPrivate[i]` receives particles moving to a nearby cell i: no atomic operation required.
- `particlesNextShared[i]` receives particles moving to a remote cell i: atomic push used.
- `particles[i]` at the next time step is obtained by merging the two.

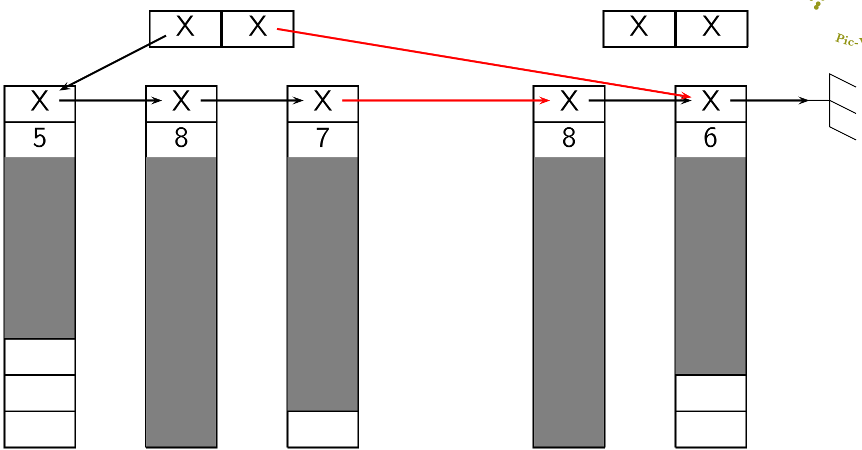
Chunk Bags: Merge Operation



Chunk Bags: Merge Operation



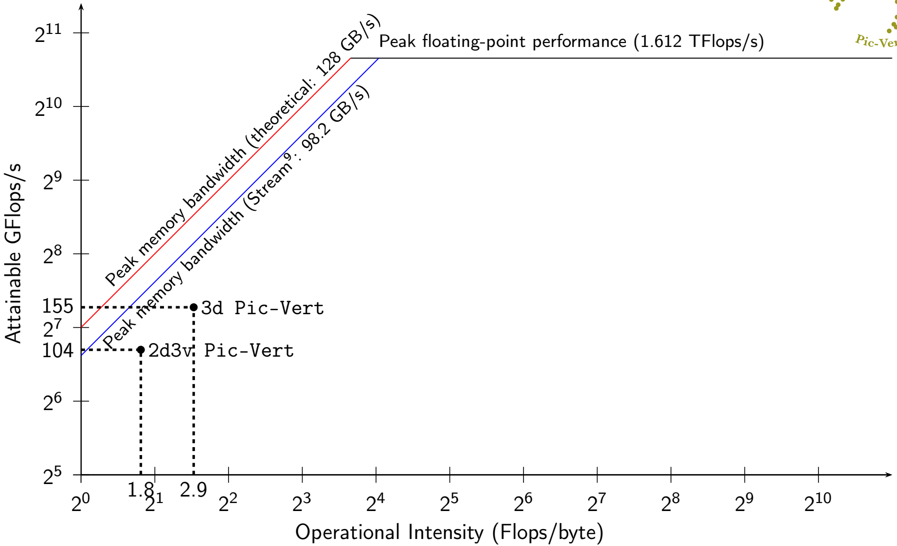
Chunk Bags: Merge Operation



Upper bound on the number of chunks: $\lceil N/K \rceil + 4 \cdot \text{nbCells}$.

All chunks allocated at initialization (no dynamic `malloc`/`free`).

Roofline Model[10] on Intel Skylake (24 Cores)



[9] McCalpin (1995) - Code v5.10 (2013)
[10] Williams, Waterman, & Patterson (2009)

Experiments with different particle velocities:

- up to 4.4% of "fast-moving particles" (more than 2 cells away),
- up to 3.7% of possible conflicts[11],
- if processed sequentially: 85% slowdown on 24 cores[12],
- when processed with our shared bags: only 7.0% slowdown.

[11] Not all fast-moving particles go out of the "extended tile" — consider a particle on the far left of the tile moving 3 cells to the right.

[12] Let t denote the single-core execution time. Assume 3.7% of sequential execution, and 96.3% using 24 cores. The parallel execution time is: $0.037t + 0.963t/24 = 1.85t/24$.

Comparison of Pic-Vert to Other Implementations

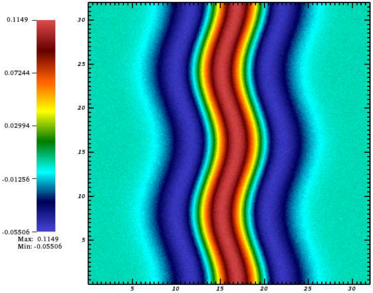
Different implementations on different architectures: cores, memory bandwidth in GB/s, number of particles processed by second (absolute and normalized w.r.t. memory bandwidth). Top: CPUs. Bottom: accelerators (GPUs, MICs).

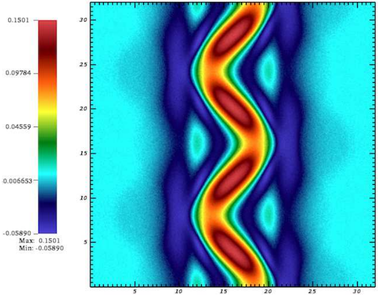
Implem.	Architecture	Cores	M.B.	Part./s	Norm.
VPIC	IBM PowerXCell 8i	9	204.8	$173 \cdot 10^6$	0.85
OSIRIS	Intel Xeon E5-2680	8	51.2	$134 \cdot 10^6$	2.62
ORB5	Intel Xeon E5-2670	8	51.2	$69 \cdot 10^6$	1.35
PICADOR	Intel Xeon E5-2697 v3	14	68	$127 \cdot 10^6$	1.87
GTC-P	Intel Xeon E5 2692 v2	12	59.7	$100 \cdot 10^6$	1.68
PIConGPU	Intel Xeon E5-2698 v3	16	68	$111 \cdot 10^6$	1.63
Pic-Vert	Intel Xeon Platinum 8160	24	128	$740 \cdot 10^6$	5.78
Pic-Vert	Intel Xeon E5-2690 v3	12	68	$374 \cdot 10^6$	5.49
PIConGPU	NVIDIA Tesla GK210	2496	480	$336 \cdot 10^6$	0.70
ORB5	NVIDIA Tesla K20X	2688	250.0	$177 \cdot 10^6$	0.71
PICADOR	Intel Xeon Phi 7250 (KNL)	68	115.2	$298 \cdot 10^6$	2.59
EMSES	Intel Xeon Phi 7250 (KNL)	68	115.2	$1300 \cdot 10^6$	11.3

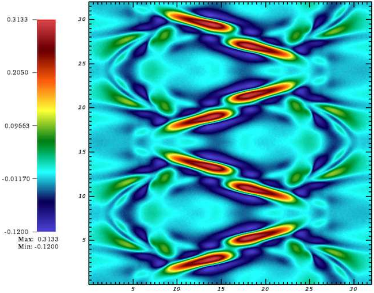
Validation: 2d3v Electron Hole Test Case[13]

- 64 billion particles,
- grid of size 512×512,
- time step 0.1,
- spatial domain $[0, 32]^2$,
- magnetic field 0.2.

Snapshots of ρ at t=0 (top right), 20 (bottom left), and 40 (bottom right).







[12] Muschietti, Roth, Carlson, & Ergun (2000)

Conclusions — Outlook

- Contributions
  - Particles sorted at all time with low memory overhead ($4 \cdot$ nbCells extra chunks, lowest in the state-of-the-art)
  - Optimal memory bandwidth usage: each particle is loaded from/written to main memory only once per iteration
  - Full advantage of SIMD
  - Efficient handling of fast particles
- Results on Intel Skylake, 24 cores, 128 GB/s
  - 740 million particles / second in 3d
  - 55% of the maximum bandwidth
- Comparison to state-of-the-art implementations
- Future outlooks
  - Solve the Maxwell equations
  - Test `Pic-Vert` on Many Integrated Core (MIC) architecture
  - Investigate the use of chunks in domain decomposition

ybarsamian@unistra.fr