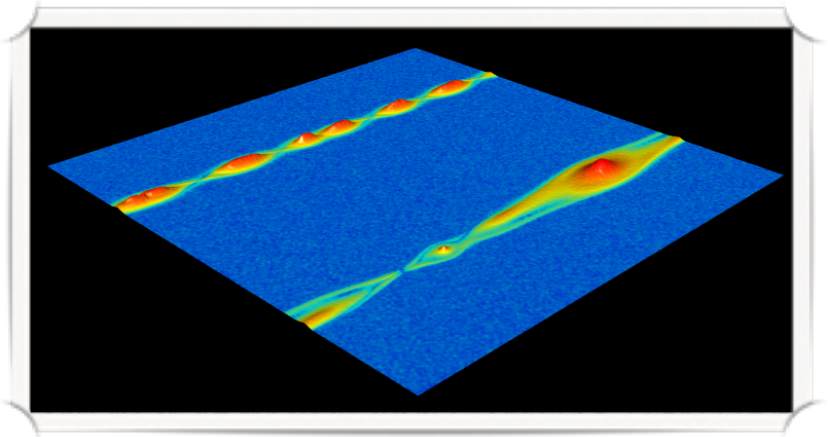
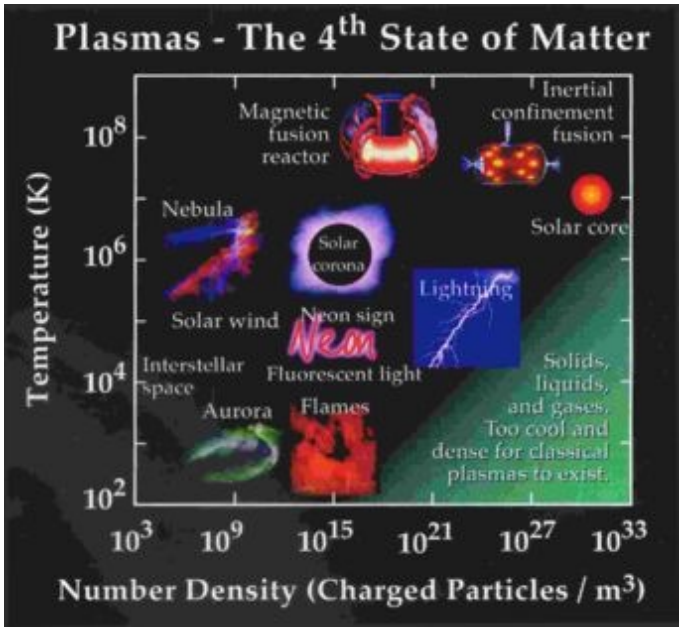
ECsim:
a massively parallel Particle-In-Cell code for plasma physics with OpenACC support

E. Boella[1], M. E. Innocenti[2], M. Bettencourt[3],
M. K. Chimeh[3], G. Lapenta[4], P. Parodi[4],
N. Shukla[5] and F. Spiga[3]

[1] Lancaster University
[2] Ruhr-Universität Bochum
[3] NVIDIA
[4] KU Leuven
[5] CINECA

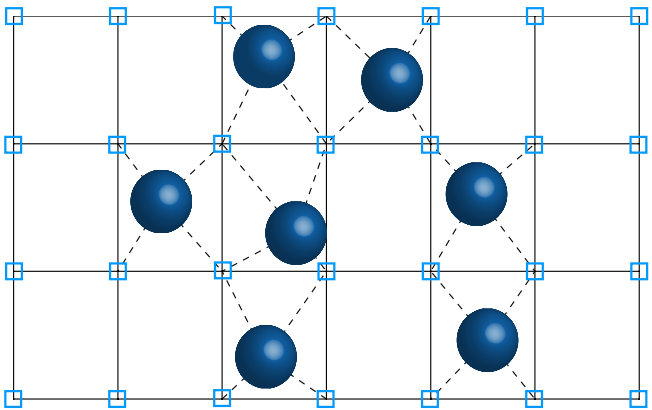


The Particle-In-Cell algorithm models the plasma microphysics



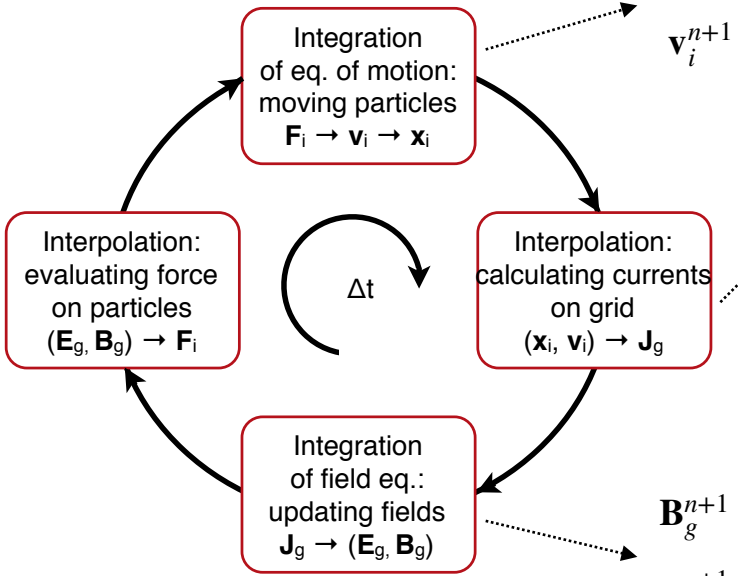
Particle-In-Cell ↔ Particle-Mesh

N_p computational particles, N_g grid cells



Dawson, Rev. Mod. Phys. 55, 403, (1983).

ECsim adopts an implicit discretisation in time for particle and field equations



$$\mathbf{x}_i^{n+1/2} = \mathbf{x}_i^{n-1/2} + \Delta t\, \mathbf{v}_i^n$$

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t \frac{q_i}{m_i}\left[\mathbf{E}^{n+\theta}(\mathbf{x}_i^{n+1/2}) + \frac{\overline{\mathbf{v}}_i \times \mathbf{B}^n(\mathbf{x}_i^{n+1/2})}{c}\right]$$

$$\mathbf{J}_g^{n+\theta} = \hat{\mathbf{J}}_g^n + \beta \sum_i q_i \alpha_i \mathbf{E}^{n+\theta}(\mathbf{x}_i^{n+1/2}) S(\mathbf{x}_i^{n+1/2} - \mathbf{x}_g) =$$

$$= \hat{\mathbf{J}}_g^n + \beta \sum_i \sum_{g'} q_i \alpha_i \mathbf{E}_{g'}^{n+\theta} S(\mathbf{x}_i^{n+1/2} - \mathbf{x}_{g'}) S(\mathbf{x}_i^{n+1/2} - \mathbf{x}_g) =$$

$$= \hat{\mathbf{J}}_g^n + \beta \sum_{g'} M_{gg'} \mathbf{E}_{g'}^{n+\theta}$$

$$\mathbf{B}_g^{n+1} = \mathbf{B}_g^n - c\Delta t \nabla_g \times \mathbf{E}^{n+\theta}$$

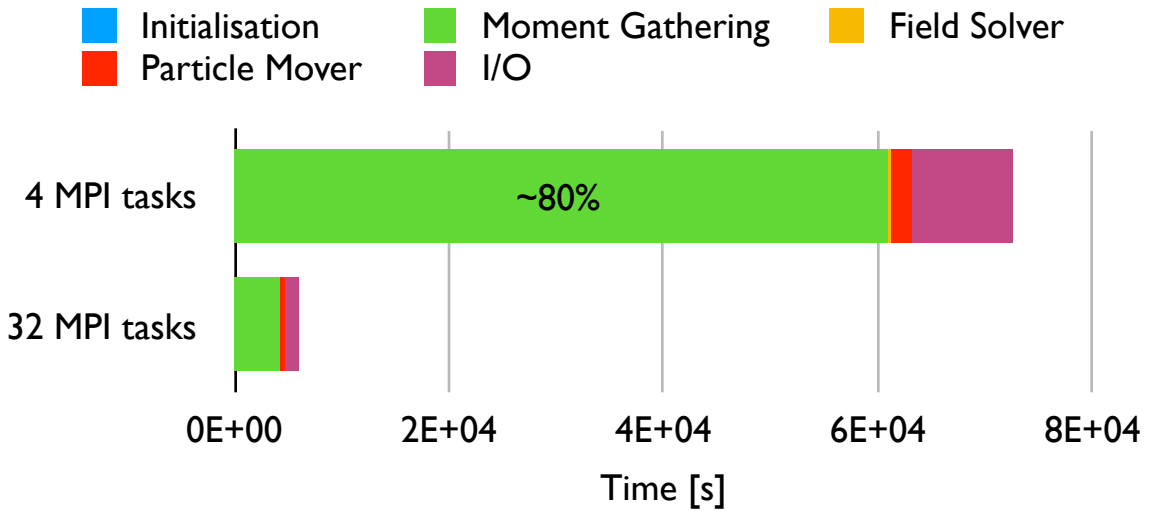
$$\mathbf{E}_g^{n+1} = \mathbf{E}_g^n + c\Delta t \left(\nabla_g \times \mathbf{B}^{n+\theta} - \frac{4\pi}{c} \mathbf{J}_g^{n+\theta}\right)$$

Lapenta et al., J. Plasma Phys. 83, 705830205 (2017). Lapenta, J. Comput. Phys. 334, 349 (2017).
Gonzalez-Herrero et al., Comp. Phys. Commun. 229, 162 (2018).

The moment gathering is the most time consuming portion of the code

ECsim

- Written in C/C++
- Parallelised with MPI
- I/O via HDF5 and H5hut
- Uses PETSc to solve fields
- Built via CMake
- Now includes OpenACC directives



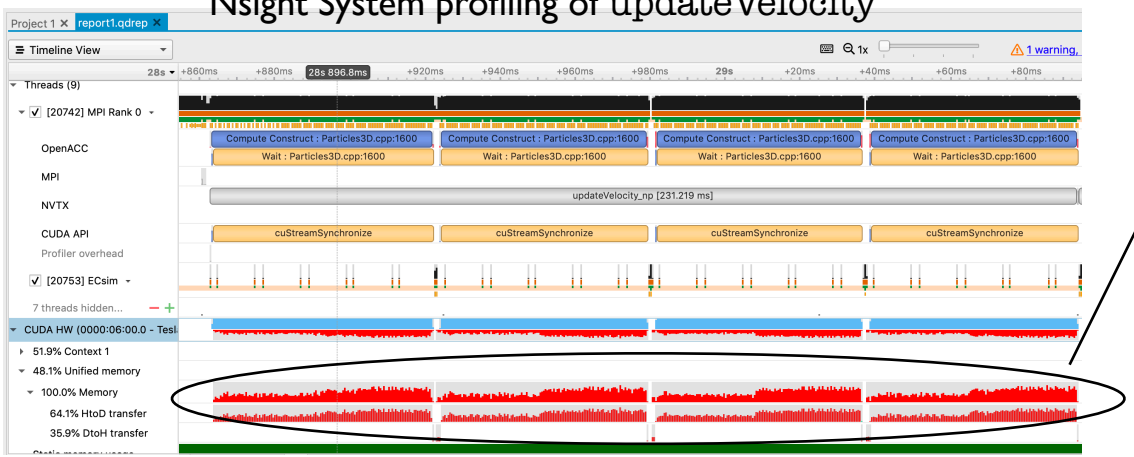
128 x 128 cells, 6400 ppc, 596 iterations
Simulations performed on Marconi100 @CINECA (Italy)
IBM Power9 32 cores/node and 4 NVIDIA V100 GPUs/node

Porting particle mover on GPU is straightforward

- updateVelocity
- updatePosition
- fixPosition

```
#pragma acc parallel loop

for (long long rest = 0; rest < nop; rest++) {

...

}
```

Nsight System profiling of updateVelocity



To solve this

```
cudaMemPrefetchAsync
(x, sizeof(double)*nop, 0, 0);
```

destination device GPU

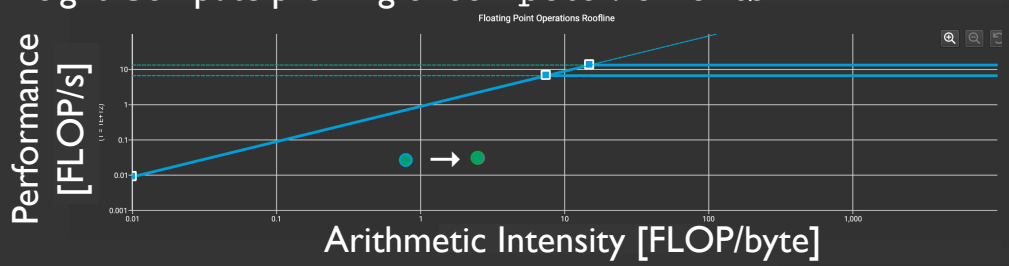
Moment gathering requires atomic operations to avoid race condition

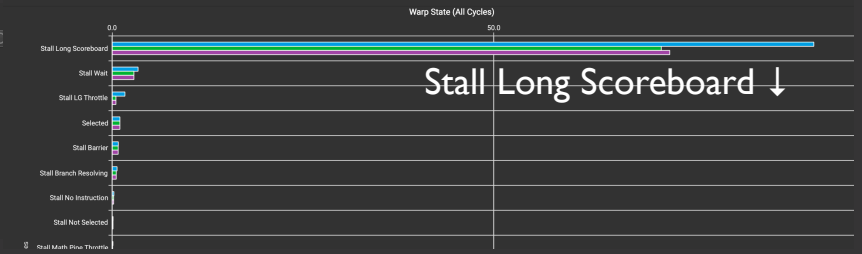
- computeMoments (Most time consuming routine of the code)

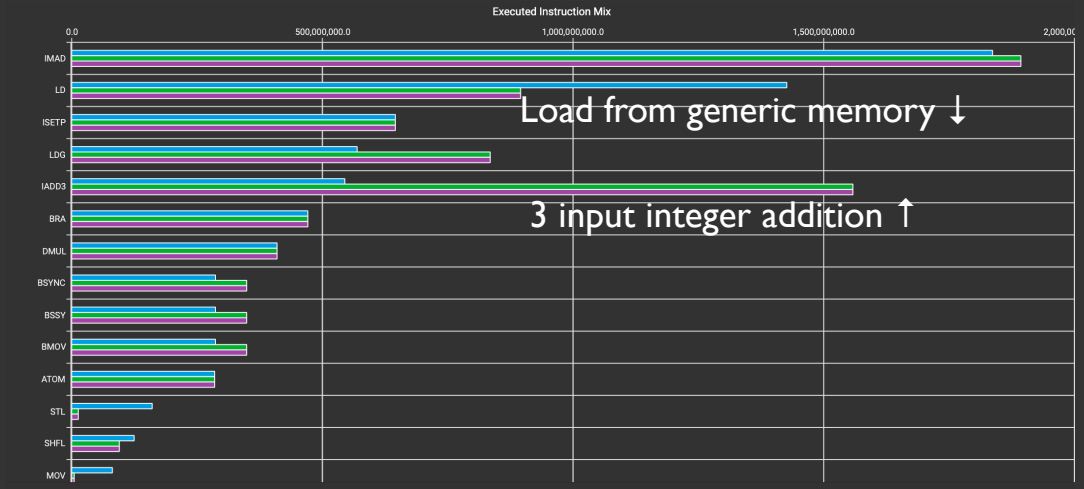
```
void EMfields3D::addRho(double weight[][2][2], int X, int Y, int Z, int is) {
 for (int i = 0; i < 2; i++)
  for (int j = 0; j < 2; j++)
   for (int k = 0; k < 2; k++) {
    const double temp = weight[i][j][k];

    #pragma acc atomic update
    rhons[is][X - i][Y - j][Z - k] += temp * invVOLn[X - i][Y - j][Z - k];
   }
}
```

We managed to improve computeMoments by increasing data locality

Nsight Compute profiling of computeMoments



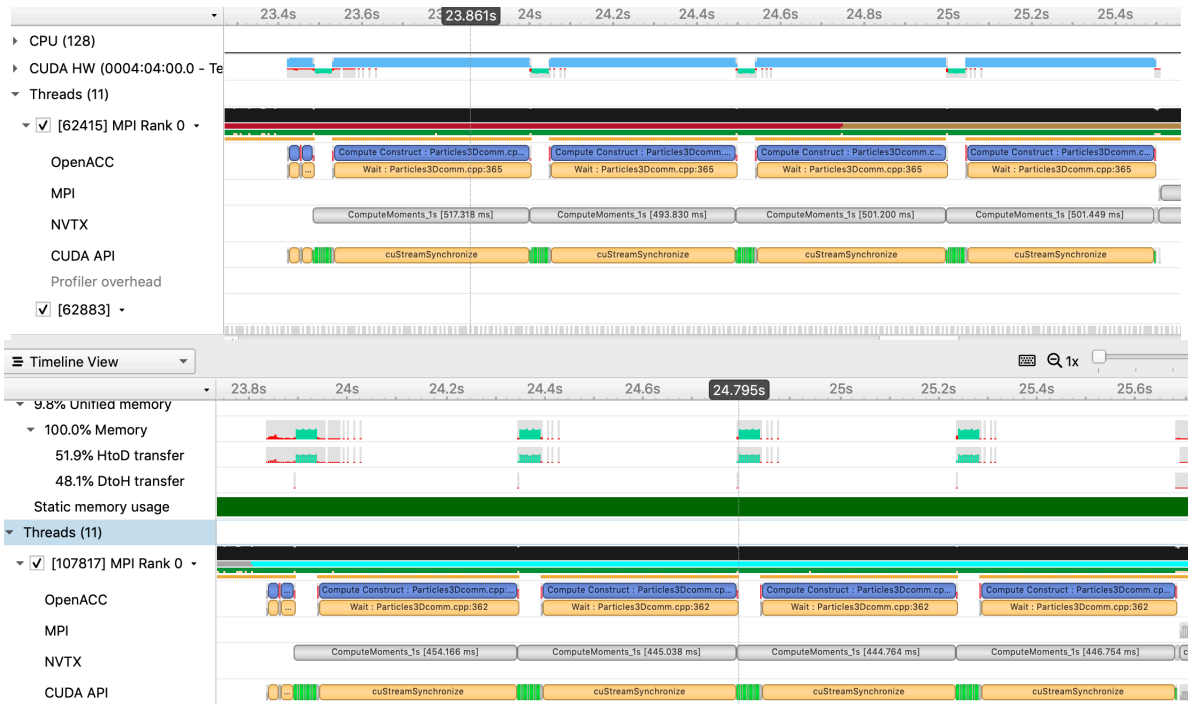




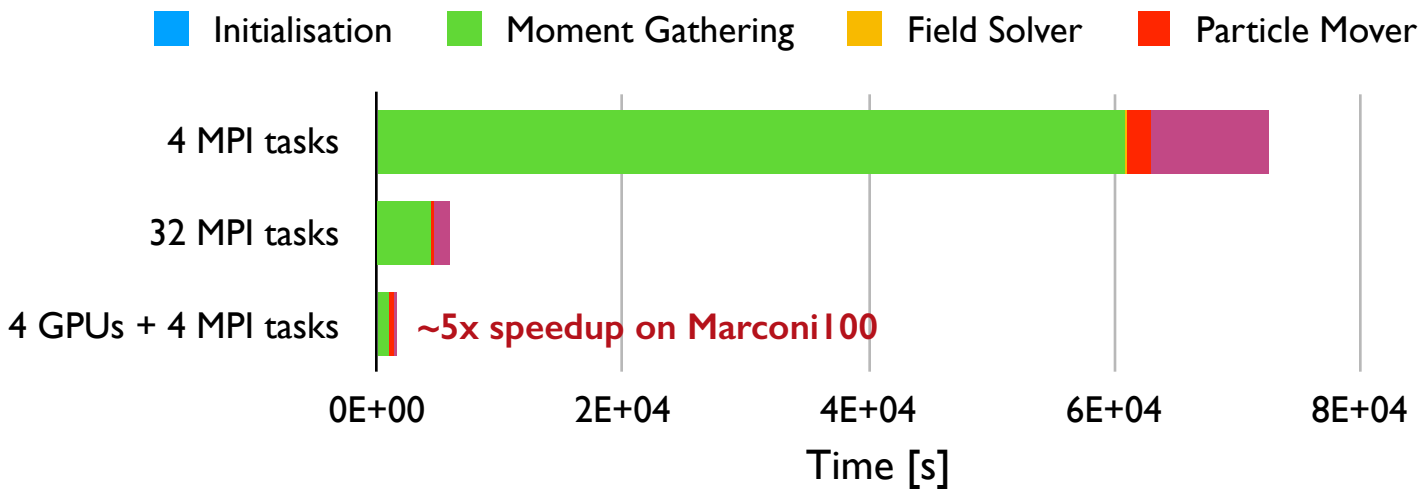
- In our simulation test, we reduced the total execution time from 1511.82 s to 1496.33 s
- We tried to go further and unrolled loops in the routine to increase data locality, but we did not get any time improvement because we increased too much the number of registers

A fine tuning optimisation led to a 12% speed up of the computeMoments kernel

Nsight System profiling of computeMoments



By offloading to GPUs the particle kernels, we achieved a 5x speedup



128 x 128 cells, 6400 ppc, 596 iterations

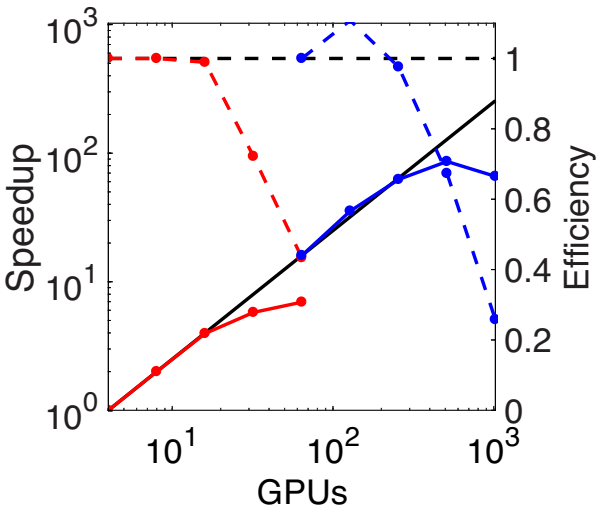
Simulations performed on Marconi100 @CINECA (Italy)

IBM Power9 32 cores/node and 4 NVIDIA V100 GPUs/node

Weak scaling shows an efficiency of 80% up to 1024 GPUs on Marconi100

Strong scaling

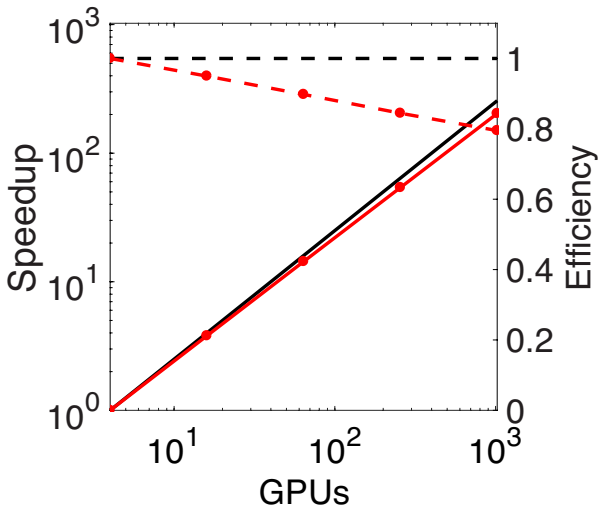
64 x 64 x 32 cells with 1296 ppc



72% efficiency up to 32 GPUs

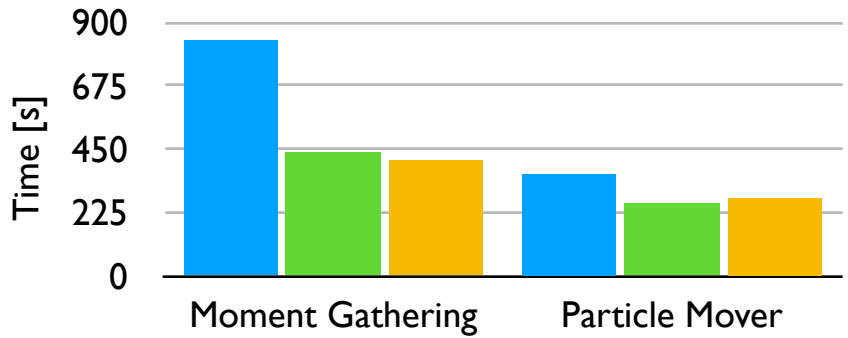
Weak scaling

128 x 128 to 2048 x 2048 cells with 6400 ppc



80% efficiency up to 1024 GPUs

On A100, Moment Gathering becomes twice as fast as on V100



Marcon100: IBM Power9 32 cores/node and 4 NVIDIA V100 GPUs/node

Juwels Booster: AMD EPYC 7402 48 cores/node and 4 NVIDIA A100 GPUs/node

MeluXina: AMD EPYC 7452 32 cores/node and 4 NVIDIA A100 GPUs/node

Summary and perspectives

The most consuming portion of ECsim on CPU is the moment gathering where particles are deposited onto the grid (~80% of the execution time).

By offloading only particle routines to GPU, a speedup of 5x was achieved.

ECsim shows an efficiency of 80% in weak scaling test up to 1024 GPUs.

Next step: porting the field solver to GPU.

This work was partially performed in the framework of the CSCS and CINECA OpenACC Hackathons. We gratefully acknowledge access to Marconi100 via ISCRA and HPCEuropa3, Juwels Booster via GCS and PRACE and MeluXina via EuroHPC.