Hybrid OpenMP/MPI Parallelization of the Charge Deposition Step in the Global Gyrokinetic Particle-In-Cell Code ORB5

Emmanuel Lanti[1]
emmanuel.lanti@epfl.ch

A. Scheinberg[1], A. Jocksch[2], N. Ohana[1], S. Brunner[1], C. Gheller[2], L. Villard[1]

[1]Ecole Polytechnique Fédérale de Lausanne (EPFL), Centre de Recherche en Physique des Plasmas, CH-1015, Lausanne, Switzerland

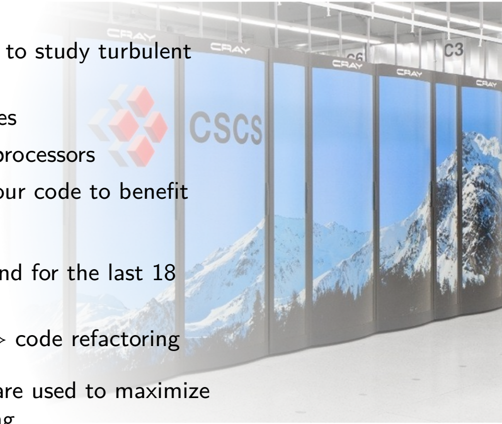
[2]CSCS, Swiss National Supercomputing Centre, Via Trevano 131, 6900 Lugano, Switzerland

PASC17 Conference
Lugano, June 27, 2017

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE | SWISS PLASMA CENTER

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Motivation



- In fusion research, gyrokinetic codes are extensively used to study turbulent transport in tokamaks
- They require an enormous amount of numerical ressources
- Top-tier HPC platforms employ many and/or multicore processors
- As computers evolve, there is a constant need to adapt our code to benefit from them

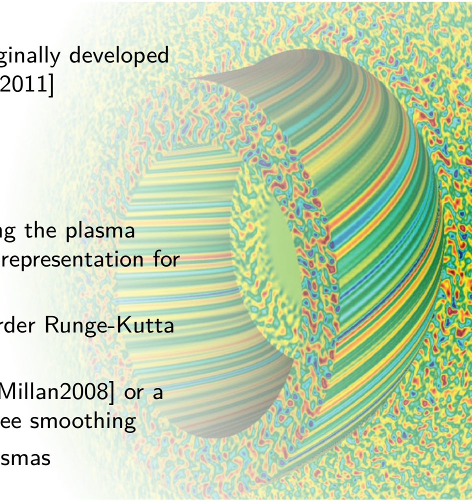
- ORB5, a gyrokinetic Particle-In-Cell code, has been around for the last 18 years (first paper in 1999)
- We don't have enough ressources to go from scratch $\Longrightarrow$ code refactoring

- In this work, different standard optimization techniques are used to maximize the time gain achievable with such a high level refactoring

Outline of the presentation

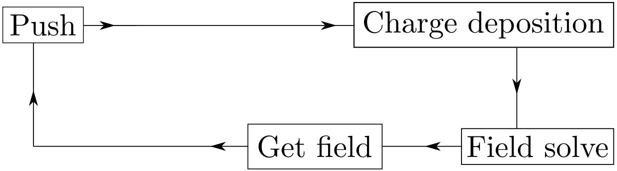
1. The global gyrokinetic ORB5 code
2. A journey towards a better performance
   - Increase data locality
   - A first try at OpenMP parallelization
   - Avoid indirect addressing
   - Avoid race conditions using colors
3. Conclusions and outlook

The global gyrokinetic code ORB5



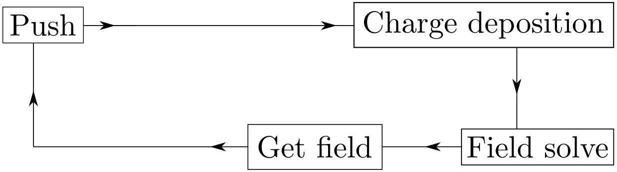
- ORB5 is a global gyrokinetic Particle-In-Cell (PIC) code originally developed at the Swiss Plasma Center [Tran1999, Jolliet2007, Bottino2011]
- It is used to describe:
  - electromagnetic (EM) turbulence of a tokamak
  - in an ideal MHD equilibrium
  - by solving the gyrokinetic equations [Brizard2007].
- It is based on the Lagrangian δf PIC scheme for representing the plasma phase space coupled with a field solver using a B-spline FE representation for solving Maxwell's equations
- The particle equations of motion are solved with a fourth order Runge-Kutta scheme
- Numerical noise is reduced using a Krook-like operator [McMillan2008] or a coarse graining procedure [Chen2007, Brunner1999], quadtree smoothing
- It handles multi-scale, multi-species, collisional, and EM plasmas

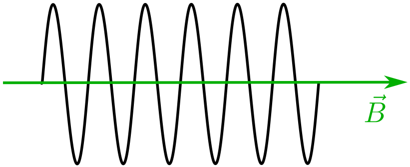
PIC vs gyrokinetic PIC



- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid

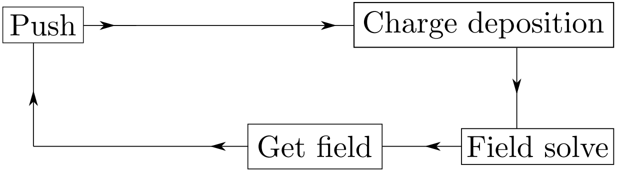
PIC vs gyrokinetic PIC

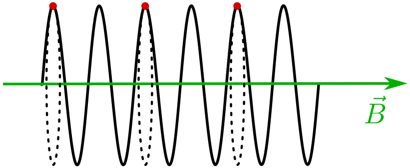




- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid

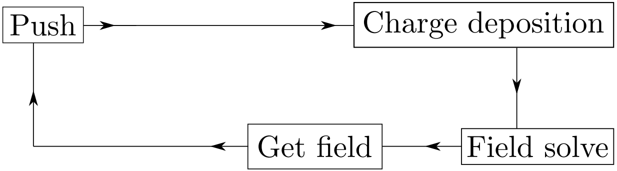
PIC vs gyrokinetic PIC

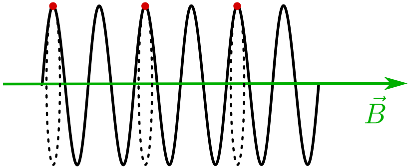




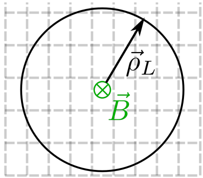
- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid

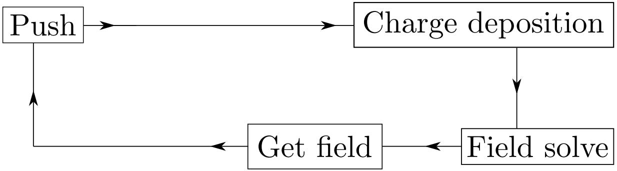
PIC vs gyrokinetic PIC

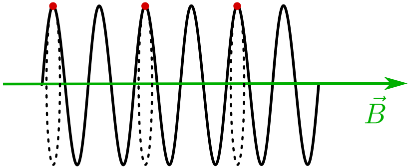




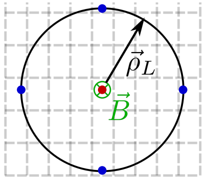
- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid



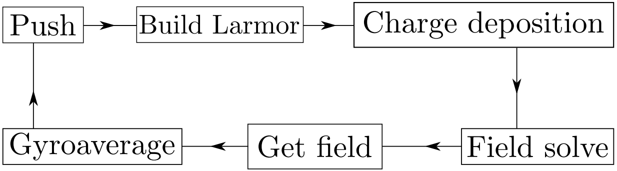


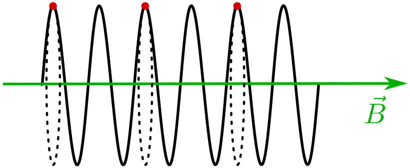


- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid

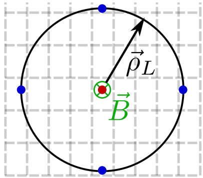


PIC vs gyrokinetic PIC



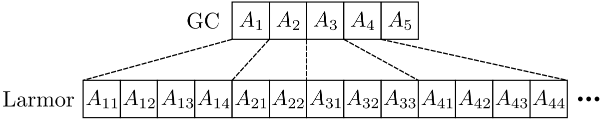


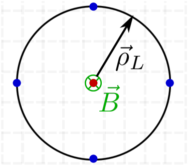
- Charge deposition is used to compute the charge/current
- Field solve compute the EM fields self-consistently with the charge/current
- Push solves for the equations of motion of the particles
- Get field interpolates the EM fields to the particle's position
- Charge deposition and get field involve interpolations from particle to field grid
- Add gyroaverage and build Larmor array operation to the PIC loop



PIC vs gyrokinetic PIC

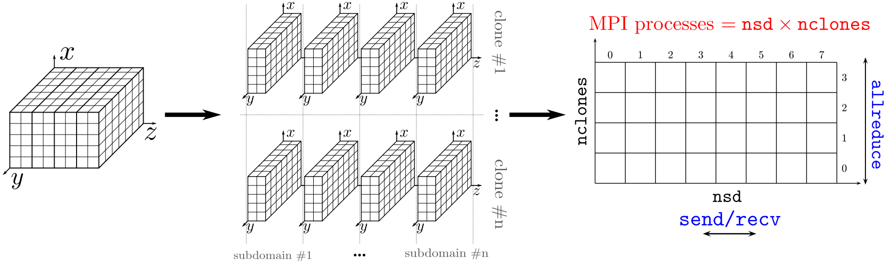
- The guiding center (GC) attributes are stored in an array
- For each GC, the number of Larmor point (LP) and their attributes are computed and stored
- As we will see, this trick allows to easily sort the LP
- However, it requires more memory !





Original ORB5 parallelization scheme

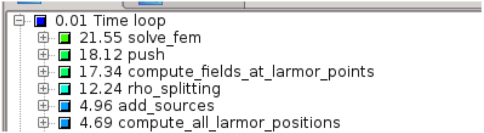
- Domain decomposition using MPI



- Showed good scalability up to several thousands of cores
- MPI communications are more and more expensive as the number of tasks increases
- A solution is to add a parallelism dimension using OpenMP to benefit from shared memory
- See next talk from A. Jocksch for a 3D domain decompostition

What are we doing in this work

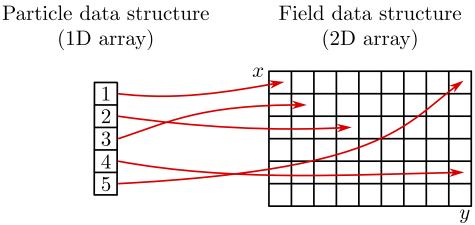
- We are trying to "optimize" ORB5, a production code, and port it to multi and manycore platforms
- We cannot start from scratch $\Longrightarrow$ incremental approach
  - No in-depth optimization
- In gyrokinetic PIC codes, the charge assignment is a critical part because:
  - it is one of the most time consuming routines



  - its parallelization is not trivial due to the **indirect assignment** (mapping of particle position to field grid)
- We will focus on the problems inherent to the charge deposition step (indirect assignment, cache reuse and vectorization) and use standard techniques to solve them
- Other parts like the push have also been treated but are presented in separate works (see A. Scheinberg's poster *PHY-03, Numerical Method Optimization in Particle-In-Cell Gyrokinetic Plasma Code ORB5* this evening)

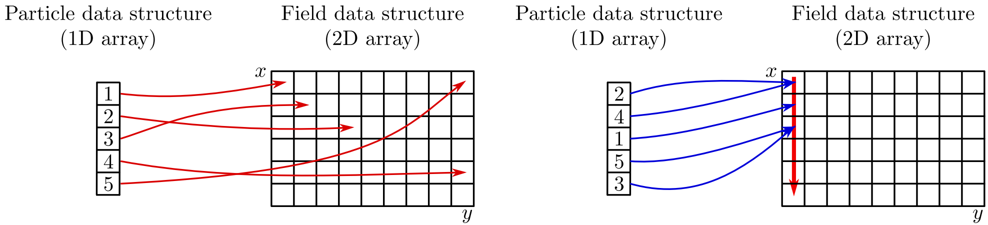
Increase data locality

- Data locality (both spatial and temporal) is a key element for a good cache reuse
- In ORB5 many operation require a mapping between particle data and field data
- Generally, nothing ensures that consecutive particles in the memory are next to each other in real space



Increase data locality

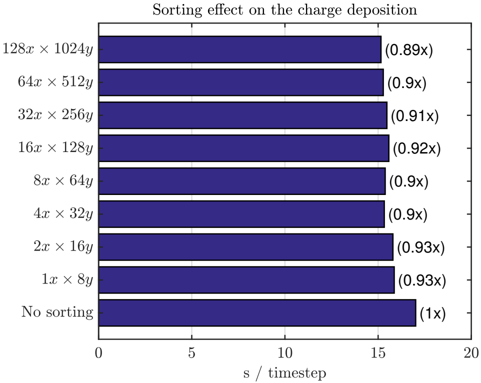
- Data locality (both spatial and temporal) is a key element for a good cache reuse
- In ORB5 many operation require a mapping between particle data and field data
- Generally, nothing ensures that consecutive particles in the memory are next to each other in real space



- However, this can be done with a particle sorting
- Counting sort implemented in ORB5 [Jocksch2016]

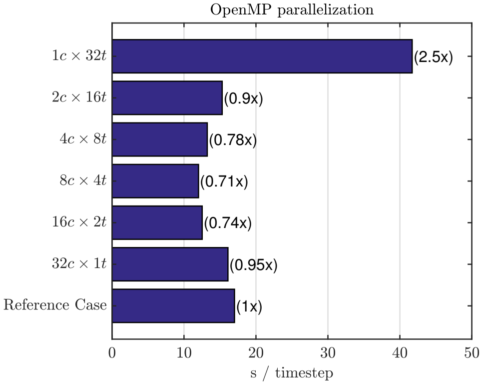
Improvement due to the particle sorting

- Test case: typical hybrid electron (TEM) run scaled down to a one node problem:
  - $128 \times 1024 \times 4$ grid
  - 8M particles (4M ions, 4M electrons)
  - 2^{nd} order B-splines
- All the timings are done on Piz Daint (XC40): 2 Intel Broadwell processors with 18 cores each
- Use Score-P profiling suite to get timings and more



- Particle sorting increases data locality and thus performance
- L1 cache misses are halved with full sorting
- Best gain with full sorting ($128s \times 1024\theta$) $\sim 10\%$
- Sorting has a cost (not shown here) !

A first try at OpenMP parallelization

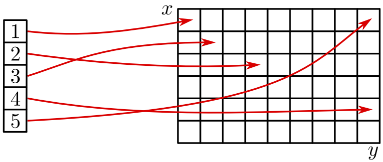


- Add OpenMP directives with private field grids and data reduction
- All the timings will be done with full sorting
- Reference case is pure MPI without sorting
- We have now 3D parallelism (MPI clones, MPI domains, and OpenMP threads)
- Vary number of clones and threads s.t. #clones $\times$ #threads $=$ #cores
- Optimal configuration: 8 clones / 4 threads
- Pure OpenMP has two problems:
  - Arrays unnecessarily allocated/deallocated
  - Load balance during reduction

Avoid indirect addressing

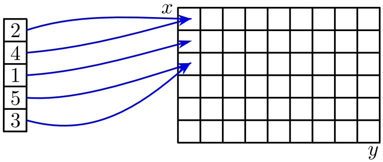
- Since PIC codes use numerical particles, it is intuitive to treat them one after the other
- The problem is that we need to map their position to the field grid:

```
do part = 1, npart
   ! Find grid-cell index
   i = x_index(part)
   j = y_index(part)
   k = z_index(part)
   array(i,j,k) = ...
end do
```

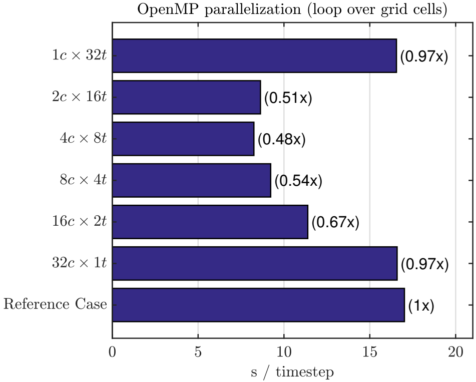


- This indirect addressing prevents auto vectorization from the compiler
- With a full sorting we can change the loop in order to avoid indirect addressing:

```
do cell = 1, ncell
   ! Grid-cell index is known
   [i, j, k] = grid_index(cell)
   do part = 1, npart_in_cell
      array(i,j,k) = ...
   end do
end do
```



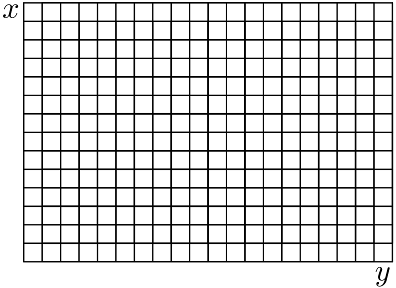
Loop over grid cells



- Reference case is pure MPI without sorting
- Optimal configuration: 4 clones / 8 threads
- Further timing decrease of 30%
- Overall performance gain due to direct addressing and vectorization

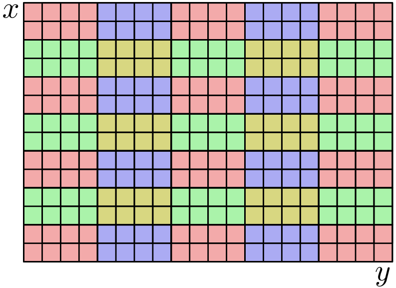
Avoid race conditions using colors

- Race conditions can be avoided using various techniques: OpenMP atomic, reduction, private data, etc
- They were tested but not very efficient as compared to the color scheme [Kong2010]



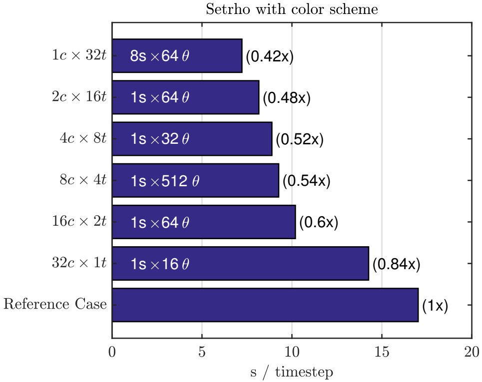
Avoid race conditions using colors

- Race conditions can be avoided using various techniques: OpenMP atomic, reduction, private data, etc
- They were tested but not very efficient as compared to the color scheme [Kong2010]



- Each color represents disjoint regions that can be treated in parallel one after the other
- Increases complexity (3 "discretizations": grid, sorting, color scheme)

Color scheme in action



- Only a 2D tilling has been implemented in ORB5
- For each configuration, all the domain tillings are tested and only the best is shown
- Reference case is pure MPI without sorting
- Now, best configuration is pure OpenMP (32 threads) with a 8×64 tilling
- Note that the color scheme was originally implemented to avoid race conditions but it also improves the load balancing

Conclusions and outlook

- Starting from its "historical" state, the ORB5 code has been cleaned and its performance has been improved with standard techniques
- Particle sorting increases data locality and improves the charge deposition step timing by $\sim 11\%$
- Adding an OpenMP layer allows to further decrease the timings by $\sim 20\%$
- Indirect addressings have been avoided by re-thinking the loops allowing to gain 30% more as compared to the "naive" OpenMP
- Finally, race conditions are avoided with a proper tilling of the field array. The best performance is a 58% timing reduction as compared to the reference case

- Some timings are still not understood. A proper profiling has to be done
- A buffered version of the color scheme is being implemented in ORB5