Numerical method optimization in particle-in-cell gyrokinetic plasma code ORB5

A. Scheinberg,* E. Lanti, N. Ohana, L. Villard, S. Brunner
Swiss Plasma Center, Ecole polytechnique fédérale de Lausanne, Switzerland
*Corresponding author. Contact: aaron.scheinberg@epfl.ch

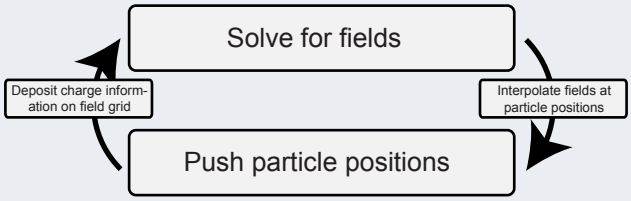
Introduction

Numerical plasma physics models improve our understanding of transport, instability growth and other poorly understood phenomena encountered in the experimental devices edging toward viable fusion energy. Computational resources must be used as efficiently as possible. Here, we examine four recent features in ORB5 and demonstrate the speedup, and thus time and resource savings, achieved by implementing these changes in the major modules of PIC gyrokinetic codes.

Model: ORB5

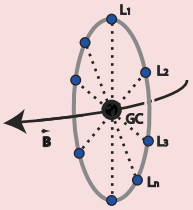
ORB5 is a particle-in-cell (PIC) delta-f gyrokinetic code. It solves for the electric (and magnetic in the case of fully electromagnetic simulations) fields on a finite-element grid using splines up to 3rd order. Additional features include noise control, collisions, and sources.

Each timestep broadly involves four steps: using particles to track the location of charge and current; using this information to solve for the electric and magnetic fields; interpolating the new fields at the particles' location; and determining the change in the particles' position and velocity due to the fields.

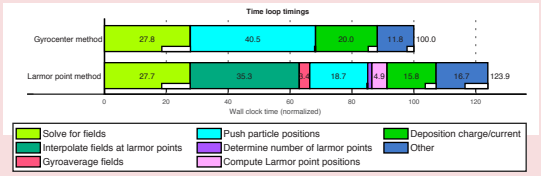


Larmor point structure

In ORB5, the "guiding center" of a particle is tracked rather than explicitly tracking the particles' gyrations around the magnetic field. (This is the gyrokinetic approximation). However, for charge deposition and field interpolation, the orbit must be sampled. Since this occurs at least twice and is computationally expensive, we introduce a "Larmor point structure" in which quantities related to the points along the orbit are stored. This restructuring actually worsens runtime, increasing it by 23.9% in our test case. However, it is necessary for the other changes.



The figure below demonstrates the difference in time spent in each module. All timings in this presentation are normalized such that the runtime of the unmodified code is 100. White insets indicate time when only one MPI process is active.



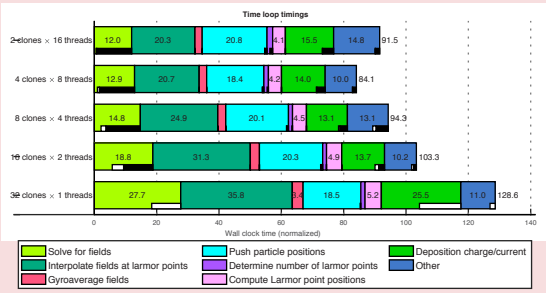
OpenMP hybrid vs. Pure MPI

Our next consideration after the structural change was introducing OpenMP multithreading. In pure MPI parallelization, ORB5 creates clones of a region for each core and distributes the particles among them, enabling division of work among the cores. Rather than data duplication, a pure OpenMP version uses shared memory while distributing the particle workload among threads. Here we use the multicore XC40 Intel Broadwell node partition of Piz Daint (CSCS). We choose how many clones/threads to use according to the formula:

(#MPI clones) x (#OpenMP threads) = (#Cores total)

Since Broadwell nodes consist of 2 sockets, using pure OpenMP (32 threads) resulted in poor performance; these runs are not shown. An optimum of 8 OpenMP threads for each of the 4 MPI clones was found to be optimal, with a runtime reduction of 15.9%.

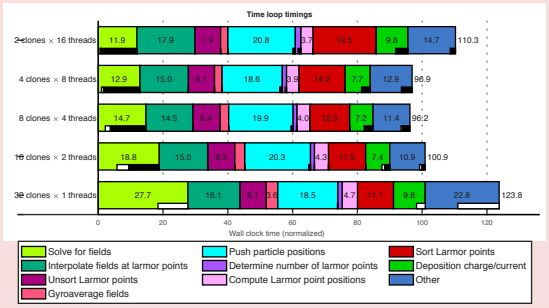
Black insets indicate time when only one OpenMP thread is active; the remaining threads are idle. This is sometimes unavoidable, but also provides directions for future improvement.



Structure of Arrays vs. Array of Structures

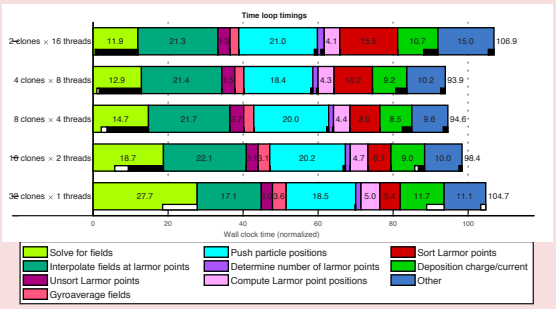
Typically, an array-of-structures (AoS) approach improves cache use, while a structure-of-arrays (SoA) approach improves SIMD efficiency. We reordered the Larmor point array's indices and compared (with sorting activated). The AoS approach (shown below) increased runtime by 19.8% in the relevant modules, and 11.7% overall. Charge deposition was particularly affected.

This result indicates that memory access is more efficient when each property (weight, position, etc.) of all particles is stored contiguously, rather than keeping all information related to a specific particle adjacent.



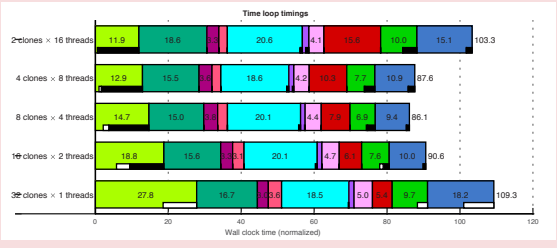
Sorting

To improve data locality, we introduced a sort of the Larmor points into bins corresponding to the grid cells on which fields are calculated. Sorting was more effective when fewer threads were present; however, for the optimum found previously (8 OpenMP threads), sorting resulted in poorer performance.



Looping by cell

Taking advantage of sorting, we can loop cell-by-cell over all the particles found in each finite-element cell. This change improved performance compared to the sorting version, but the unsorted version remained most effective.



Conclusion

Introducing OpenMP resulted in a reduction of runtime by 15.9% for the test case presented here. There is still potential for an additional ~12% gain via OpenMP optimization, particularly in the field solver.

Sorting was not found to be immediately beneficial, even with a loop-by-cell method. However, further exploitation of fully sorted Larmor points could lead to improvement. Finally, storage of particle data as structures of arrays was found to be important for optimization.

With some 1-3 million node-hours potentially used for simulations by the ORB5 community in the coming year, a 15% gain could correspond to hundreds of thousands euros saved.

Future work will evaluate multi-node runs; the improvement on GPU architecture and KNL; hyperthreading; and the efficiency gains for different test cases involving fully kinetic electrons and electromagnetic scenarios.