XGCm: An Unstructured Mesh Gyrokinetic Particle-in-cell Code for Exascale Fusion Plasma Simulations

Chonglin Zhang, Gerrett Diamond, Cameron W. Smith, Mark S. Shephard

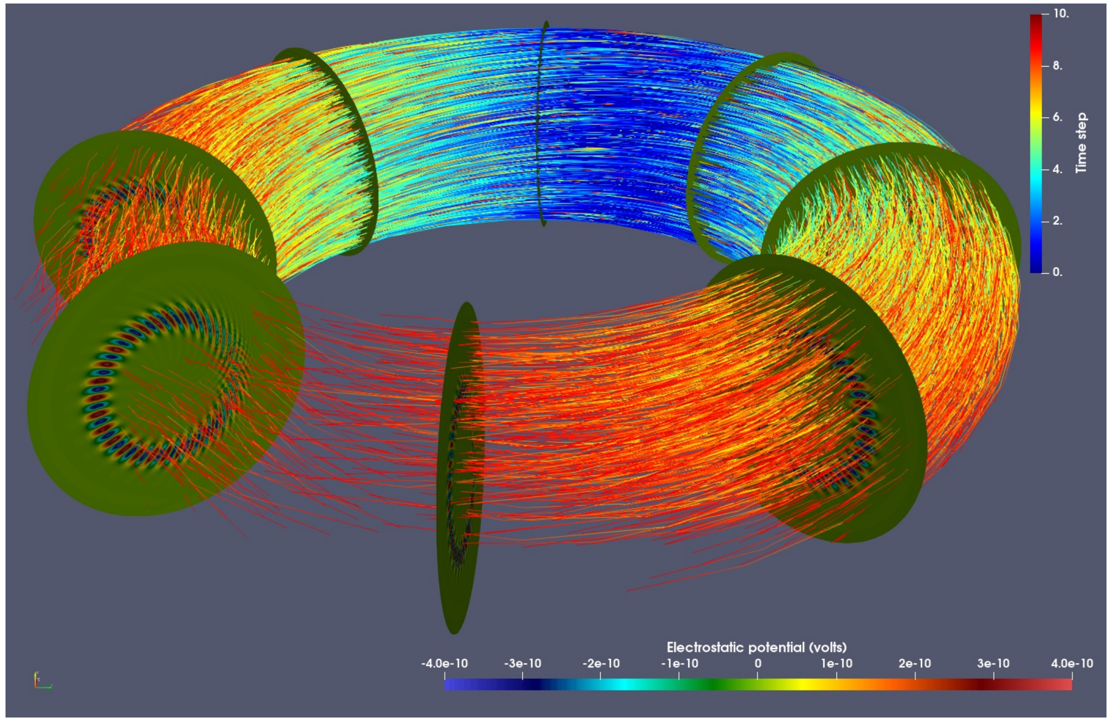
Scientific Computation Research Center

Rensselaer Polytechnic Institute

Overview

XGCm is a new gyrokinetic particle-in-cell (PIC) code for fusion plasma simulations:

- Mesh-centric approach to handle particle operations;
- Distributed unstructured mesh;
- Physical algorithms from the well-established XGC code;
- Omega_h: unstructured mesh management (Kokkos based);
- PUMIPic: particle management (Omega_h based);
- PETSc: linear equation solver;
- All operations performed on GPU (currently working on Nvidia GPUs: Summit, Perlmutter, and RPI's AiMOS; porting to AMD and Intel GPU underway);
- Aimed at exascale fusion plasma simulations.



Demonstration of the particle trajectories at several time steps, and the electrostatic potential results for the Cyclone ITG test case 5 of Burckel et. al. (2010). Particle trajectories are colored by time step number, and each poloidal plane is colored by the electrostatic potential.

Outline

The Vlasov equation and gyrokinetic particle-in-cell method

- Starting from the Boltzmann equation

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{a} \cdot \frac{\partial f}{\partial \boldsymbol{v}} = (\frac{\partial f}{\partial t})_{coll}$$

- Long-range Coulomb interaction
- Collisionless Boltzmann equation

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{a} \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0$$

Vlasov Equation

$$\frac{\partial f_s}{\partial t} + \boldsymbol{v}_s \cdot \frac{\partial f_s}{\partial \boldsymbol{x}} + \frac{q_s}{m_s}(\boldsymbol{E} + \boldsymbol{v}\times\boldsymbol{B}) \cdot \frac{\partial f_s}{\partial \boldsymbol{v}} = 0$$

Self-consistent electric field $\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon}$

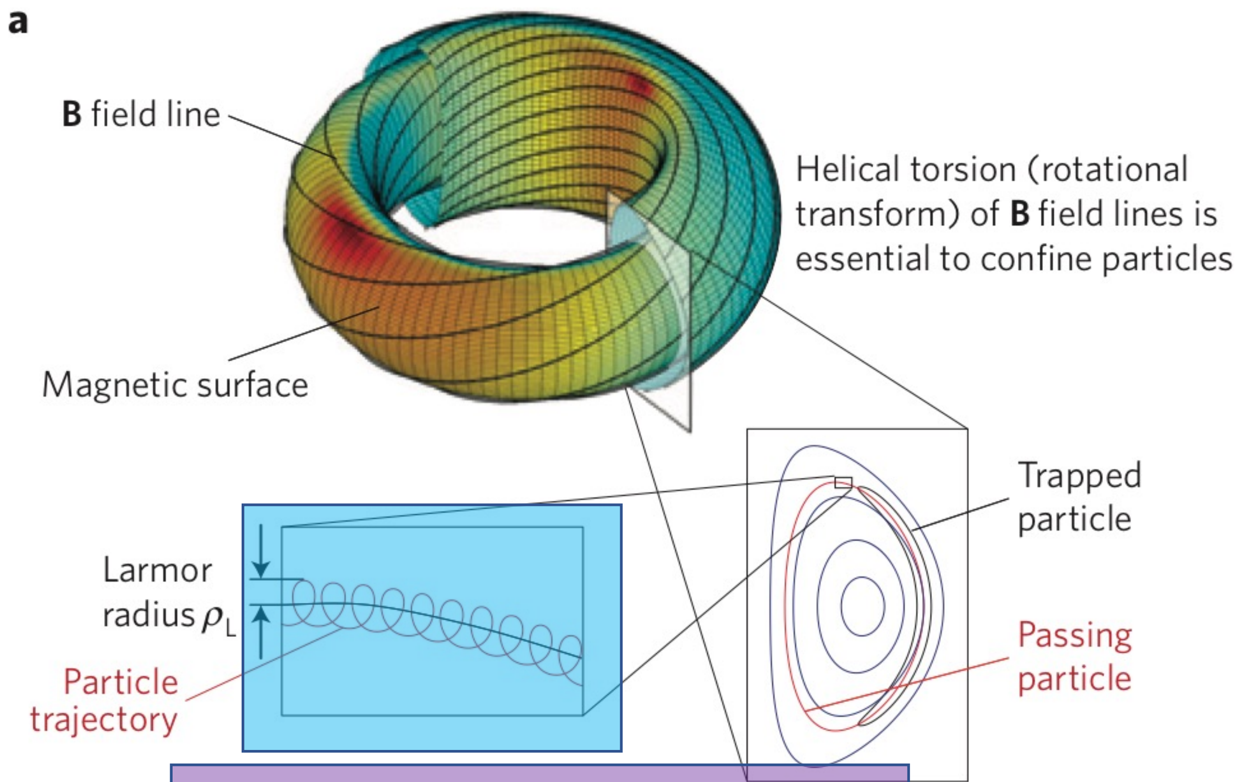
Lorentz force on charged particles

$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B})$, or $\boldsymbol{a} = \frac{q}{m}(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B})$,

$\boldsymbol{E}$: electric field, $\boldsymbol{B}$: magnetic field

The Vlasov equation and gyrokinetic particle-in-cell method

- Particle-in-cell method numerically solves the Vlasov equation;
- Monte Carlo particle method;
- Each particle represents a large amount of (identical) real charged ions/electrons;
- Main operations in the PIC method:
  - Particle push;
  - Particle charge deposition;
  - Field solve;
  - Field to particle interpolation.
- Gyrokinetic particle-in-cell method:
  - Particle guiding-center position, instead of actual particle position is simulated.



∂f_s ∂f_s q_s ∂f_s 0

Example magnetic field and particle trajectories[1].

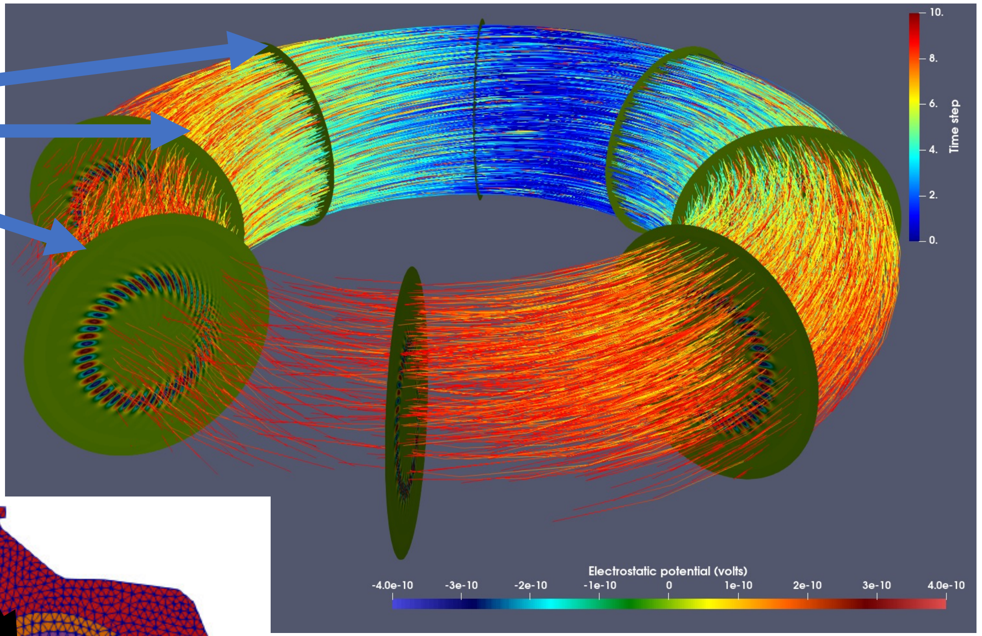
Self-consistent electric field $\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon}$

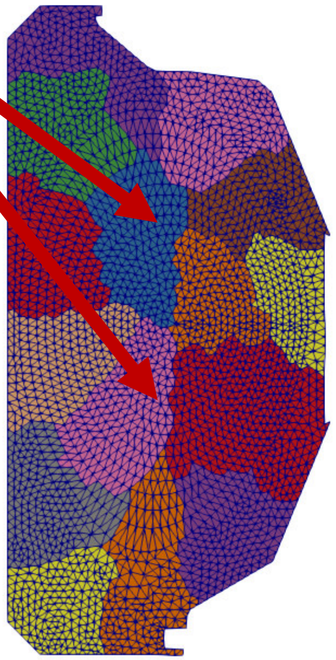
[1] A. Fasoli, S. Brunner, W. A. Cooper, J. P. Graves, P. Ricci, O. Sauter and L. Villard, *Nature Physics*, vol 12, pp 411–423, 2016.

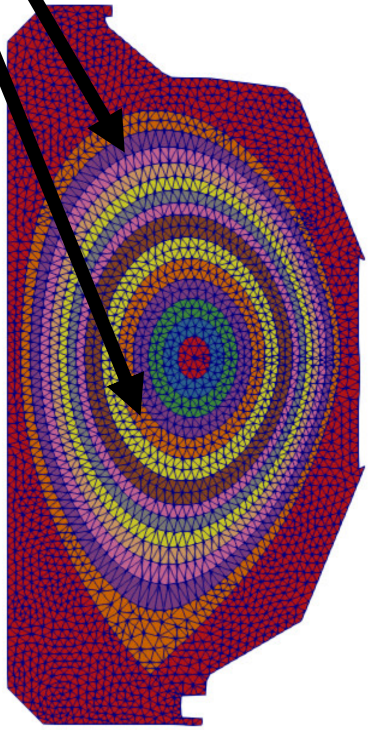
Unstructured partitioned mesh: Omega_h

Three levels of mesh partitions:

- Toroidal partition: partition particles into different MPI ranks along the torus direction;
- Poloidal partition: in each poloidal plane, mesh is partitioned into different MPI ranks according to the flux curves; particles are associated with each triangle element of the distributed mesh;
- Solver partition: a (potentially) different mesh partition in each poloidal plane for solving the gyrokinetic Poisson equation.



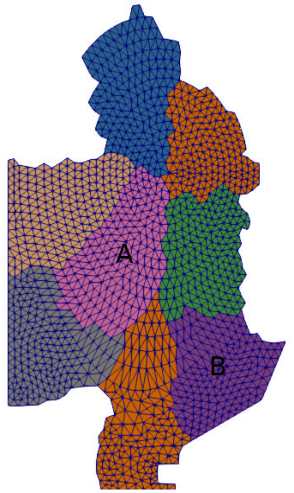




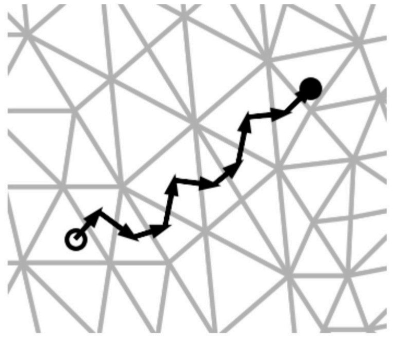
As a first step
solver partition = polc

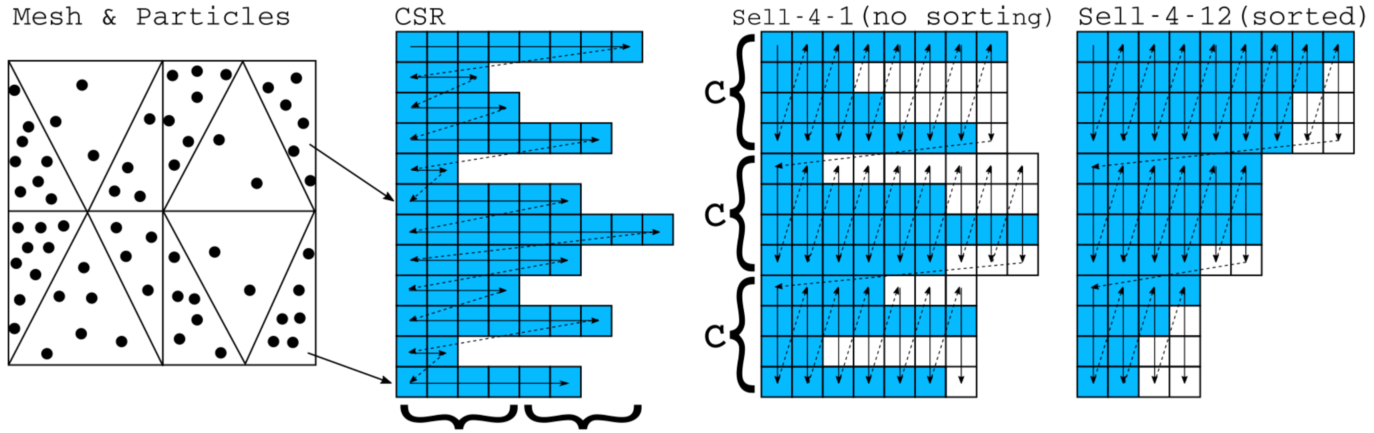
Particle management in distributed unstructured mesh: PUMIPic

- A distributed mesh approach for PIC that can scale both the particles and mesh and is performant.
- Particles accessed through mesh to support faster data access and mesh/particle operations;
- Mesh distribution based on core parts and buffer parts to ensure on-process movement of particles in a particle push operation;
- Mesh infrastructure to support all major PIC operations on GPU: particle, particle-mesh, mesh, mesh-mesh operations;



PICpart generated for the core part A using 4 layers of breadth-first traversal.


Path of a particle through a 2D triangular mesh using edge adjacencies



The storage of particles in a set of mesh elements (left) in a CSR (middle) and two SCS (right) with no sorting and with sorting. Arrows on each structure show the continuous layout of memory.

[1] Gerrett Diamond, Cameron W.Smith, Chonglin Zhang, Eisung Yoon, and Mark S. Shephard, *PUMIPic: A mesh-based approach to unstructured mesh Particle-In-Cell on GPUs,* Journal of Parallel and Distributed Computing, Vol 157, pp 1-12 (2021).

Solving electrostatic gyrokinetic Poisson equation using PETSc

Equations being solved (long wavelength limit)[1]

$$-\nabla_\perp \cdot \frac{n_i m}{eB^2}\nabla_\perp \boxed{\Phi} + n_0\frac{\delta\Phi}{T_e} = (\bar{n}_i - \delta n_e)$$

Guiding center density $\bar{n}_i = n_{i,0} + \delta\bar{n}_i = n_0 + \delta\bar{n}_i$

$$\bar{n}_i = \frac{1}{2\pi}\int f_i(\mathbf{X},\mu,u)\delta(\mathbf{X} - x + \vec{\rho}_i)d\mathbf{X}d\mu d\alpha$$

Field (right-hand-side and solution vectors) storage:

- If solve on GPU, data stays on GPU memory:
  - GPU Read/write: Omegah::Write<o::Real>
  - GPU Read only: Omegah::Reals
- If solve on CPU, data copied to CPU memory:
  - CPU Read/write: Omegah::HostWrite<o::Real>
  - CPU Read only: Omegah::HostRead<o::Real>

The solution process:

- After particle push, obtain right-hand-side vector (charge density) resulting from charge scatter;
- Copy right-hand-side vector from Omega_h to PETSc;
- Solve either on GPU or CPU with PETSc;
  - ❑ Linear matrix is assembled once on GPU.
- After PETSc solve, copy solution vector from PETSc to Omega_h;
- Calculate electric field from solution vector and perform particle push;
- Repeat the above process.

[1] S. Ku, C. S. Chang, R. Hager, R. M. Churchill, G. R. Tynan, I. Cziegler, M. Greenwald, J. Hughes, S. E. Parker, M. F. Adams, E. D'Azevedo, and P. Worley , *A fast low-to-high confinement mode bifurcation dynamics in the boundary-plasma gyrokinetic code XGC1,* Physics of Plasmas 25, 056107 (2018)

Solving electrostatic gyrokinetic Poisson equation using PETSc

```
  // copy xgcm field into petsc vector
  template<class Device>
  PetscErrorCode Poisson<Device>::copyVec_xgcm_to_petsc(o::Reals& xgc_vec,
                                                         Vec petsc_vec) {

    assert(xgc_vec.size() == simmesh->nverts());
    Vec bloc;
    PetscErrorCode ierr = DMGetLocalVector(dm, &bloc); CHKERRQ(ierr);
    PetscScalar *bwrite;

    // handle GPU or CPU solve
#ifdef XGCM_GPU_SOLVE
    ierr = VecCUDAGetArrayWrite(bloc, &bwrite); CHKERRQ(ierr);
    const auto p2lv = partVtx_to_locVec;
    auto set_petsc_vec = OMEGA_H_LAMBDA(const o::LO vtx) {
      const auto vecIdx = p2lv[vtx];
      if (vecIdx >= 0) {
        bwrite[vecIdx] = xgc_vec[vtx];
      }
    };
    o::parallel_for(simmesh->nverts(), set_petsc_vec, "set_petsc_vec");
    ierr = VecCUDARestoreArrayWrite(bloc, &bwrite); CHKERRQ(ierr);
```

If solve on GPU

If Solve on GPU, specify PETSc matrix and vector as

- -dm_vec_type cuda
- -dm_mat_type aijcusparse

2. Call PETSc to solve the linear equation

```
      ierr = KSPSolve(ksp, b, u); CHKERRQ(ierr);
      ierr = KSPGetSolution(ksp, &u);CHKERRQ(ierr);
```

3. After solve, copy solution vector to XGCm

```
      // scatter solution vector u to xgc field
      ierr = copyVec_petsc_to_xgcm(u, pot); CHKERRQ(ierr);
```

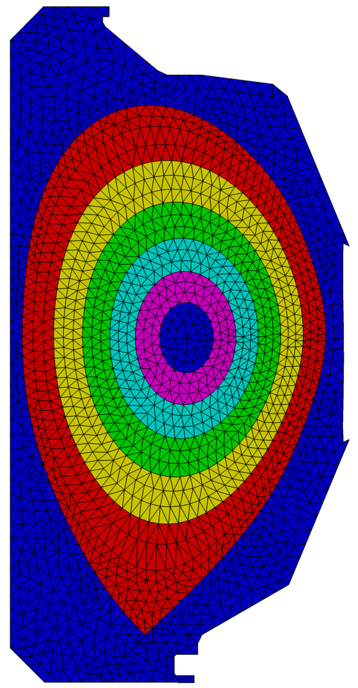
```
#else
    o::HostRead<o::LO> p2lv(partVtx_to_locVec);
    o::HostRead<o::Real> xgc_vec_host(xgc_vec);
    ierr = VecGetArrayWrite(bloc, &bwrite); CHKERRQ(ierr);
    for (int vtx = 0; vtx < xgc_vec.size(); vtx++) {
      const o::LO vecIdx = p2lv[vtx];
      if (vecIdx >= 0) {
        bwrite[vecIdx] = xgc_vec_host[vtx];
      }
    }
    ierr = VecRestoreArrayWrite(bloc, &bwrite); CHKERRQ(ierr);
#endif
    ierr = DMLocalToGlobal(dm, bloc, INSERT_VALUES, petsc_vec); CHKERRQ(ierr);
    ierr = DMRestoreLocalVector(dm, &bloc); CHKERRQ(ierr);

    PetscFunctionReturn(0);
  }
```

If solve on CPU

Solving electrostatic gyrokinetic Poisson equation using PETSc

Solver partition

- Currently used the same partition as the poloidal partition;
- Easiest;
- More importantly, big time cost to support a different solver partition.

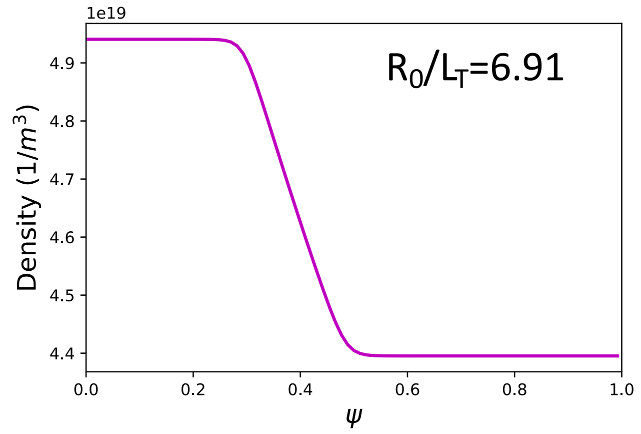


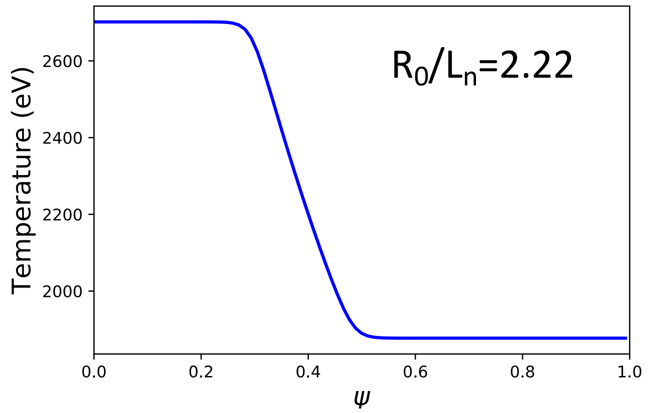
KSPSolve time comparison (in unit of seconds)

# of mesh partitions	# of mesh triangles per rank (in thousands)	Solve on GPU	Solve on CPU
1	2400	8.181	247.93
6	400	10.187	51.552
12	200	9.6197	22.622
24	100	10.839	
48	50	14.418	
96	25	11.876	

- -ksp_type cg
- -pc_type gamg
- -mg_levels_ksp_type chebyshev
- -mg_levels_pc_type jacobi

Code validation: cyclone base case with circular geometry

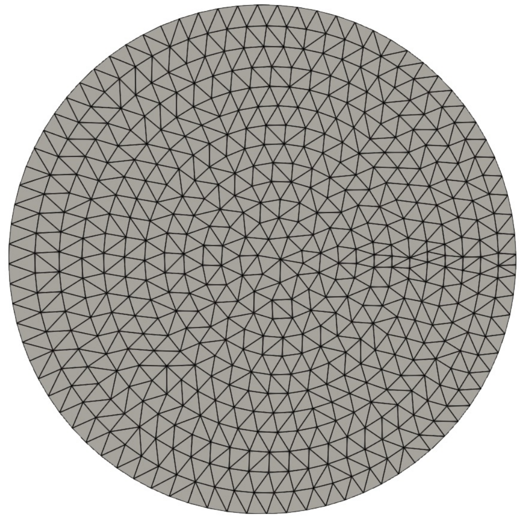




Background density and temperature profile, ψ is normalized poloidal magnetic flux as in the left figure[1].

Simulation setup

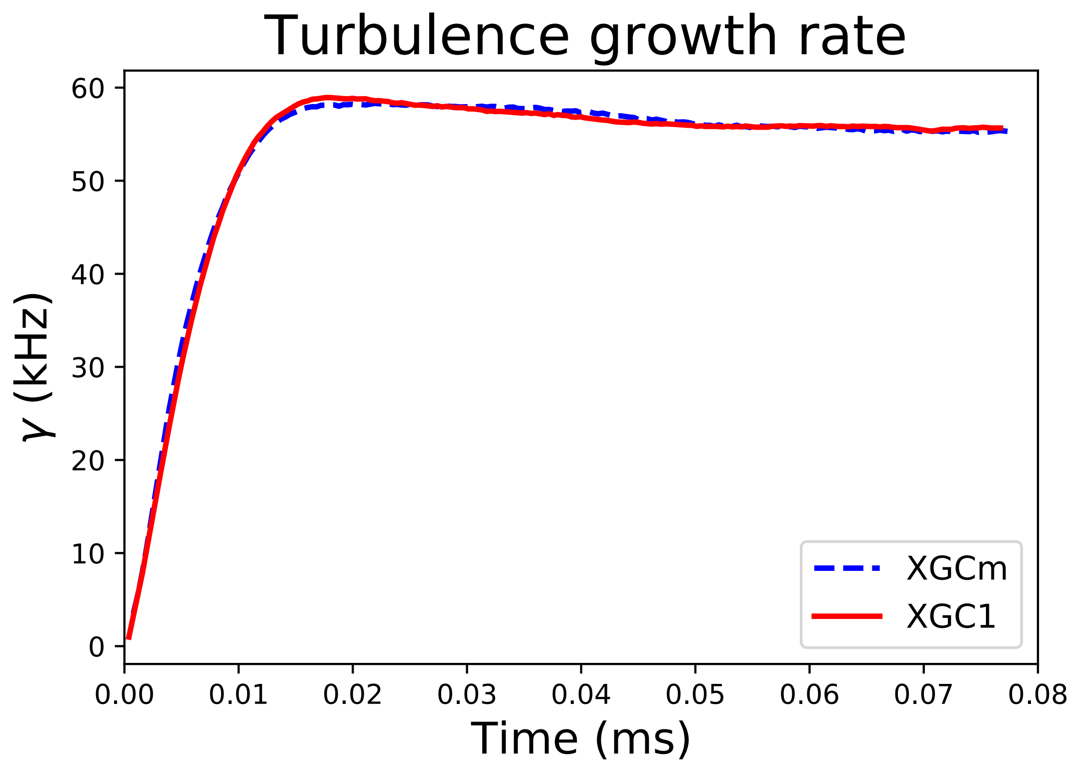
- Mesh size: 590,143 mesh triangles, 296,046 mesh vertices;
- 8 poloidal planes/partitions, or 8 GPUs;
- 20 million particles per GPU;
- An initially perturbed density field is used, corresponding to a single toroidal mode number n=24, with Gaussian shape in both the radial and poloidal directions;
- dt = 3.91e-7 second;
- Simulation was run for 200 time steps.



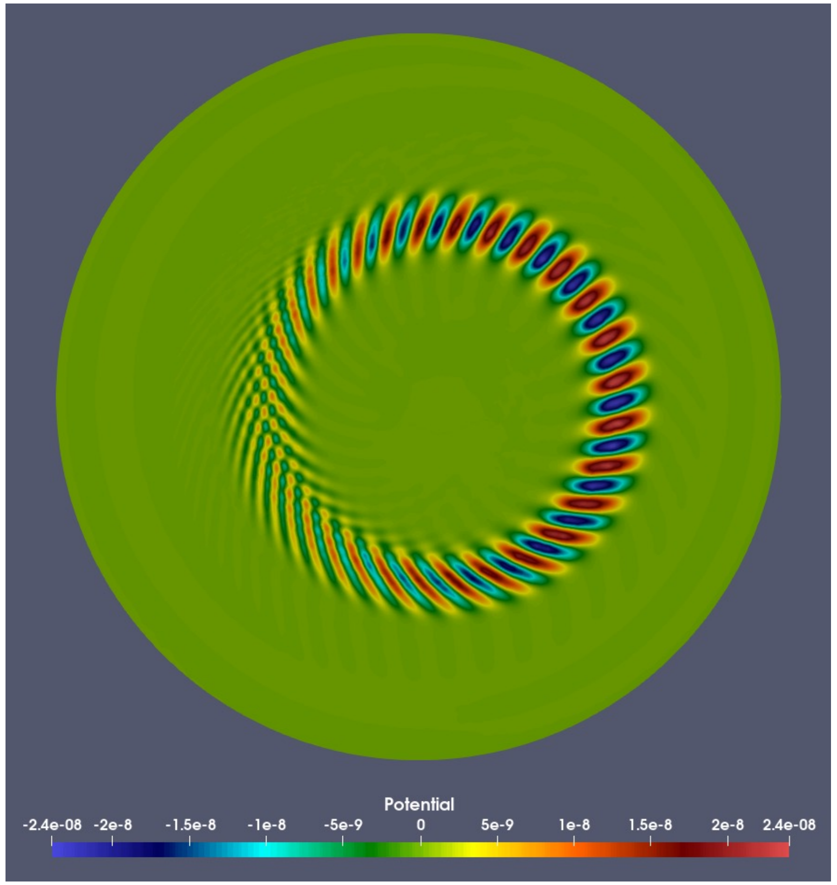
Coarse mesh is shown here for visualization

[1] G. Merlo, J. Dominski, A. Bhattacharjee, C. S. Chang, F. Jenko, S. Ku, E. Lanti, and S. Parker, *Cross-verification of the global gyrokinetic codes GENE and XGC,* Physics of Plasmas 25, 062308 (2018)

Code validation: cyclone base case with circular geometry



Growth rate, γ, of the turbulent electrostatic potential over time. Here, $\gamma = \frac{d log(\varphi(t))}{dt}$, with $\varphi(t)$ is the square-averaged turbulent electrostatic potential at time t, $\log()$ is the logarithm function, and $\frac{d}{dt}$ is the time derivative.



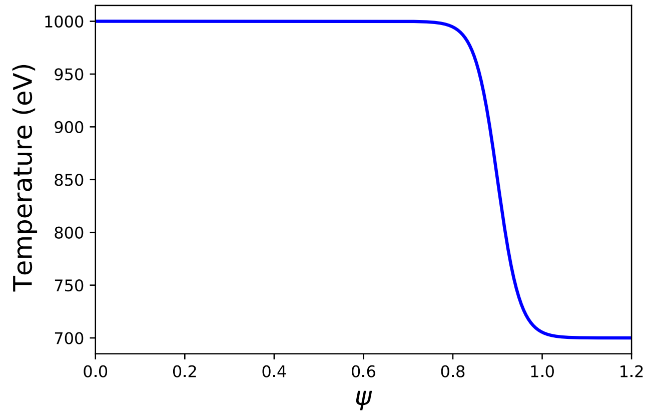
Contour plot of turbulent electrostatic potential on one poloidal plane at time step 200.

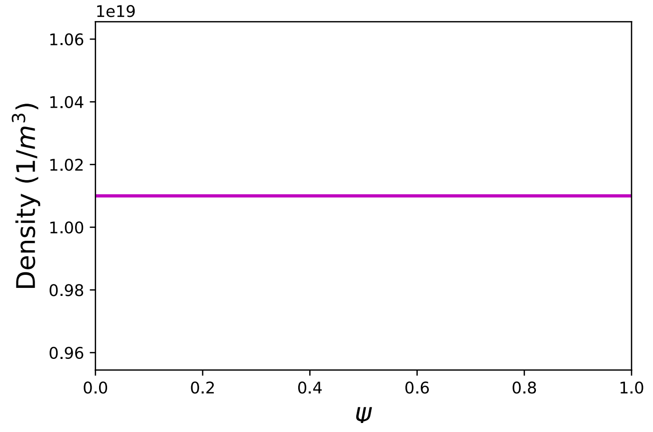
Ion Temperature Gradient (ITG) with DIII-D geometry

- Adiabatic electron;
- DIII-D equilibrium 096333;

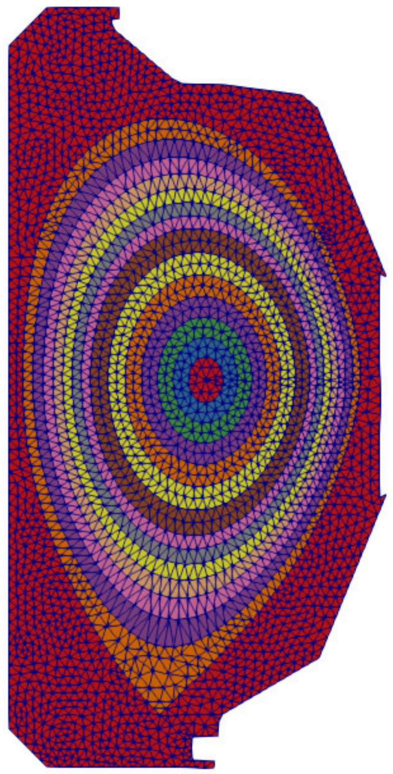
Simulation details

- 16 poloidal planes; each with 20 million particles;
- Each poloidal plane has 400,276 triangle elements;
- Simulations were run for 1000 ion time steps;
- dt = 3.13e-7 second

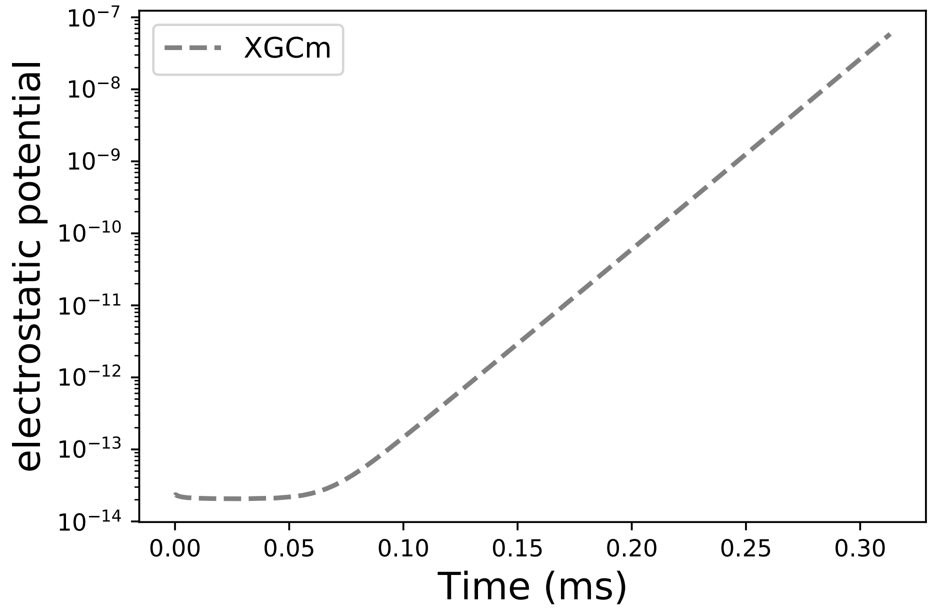




Initial background density and temperature profile, ψ is normalized poloidal magnetic flux

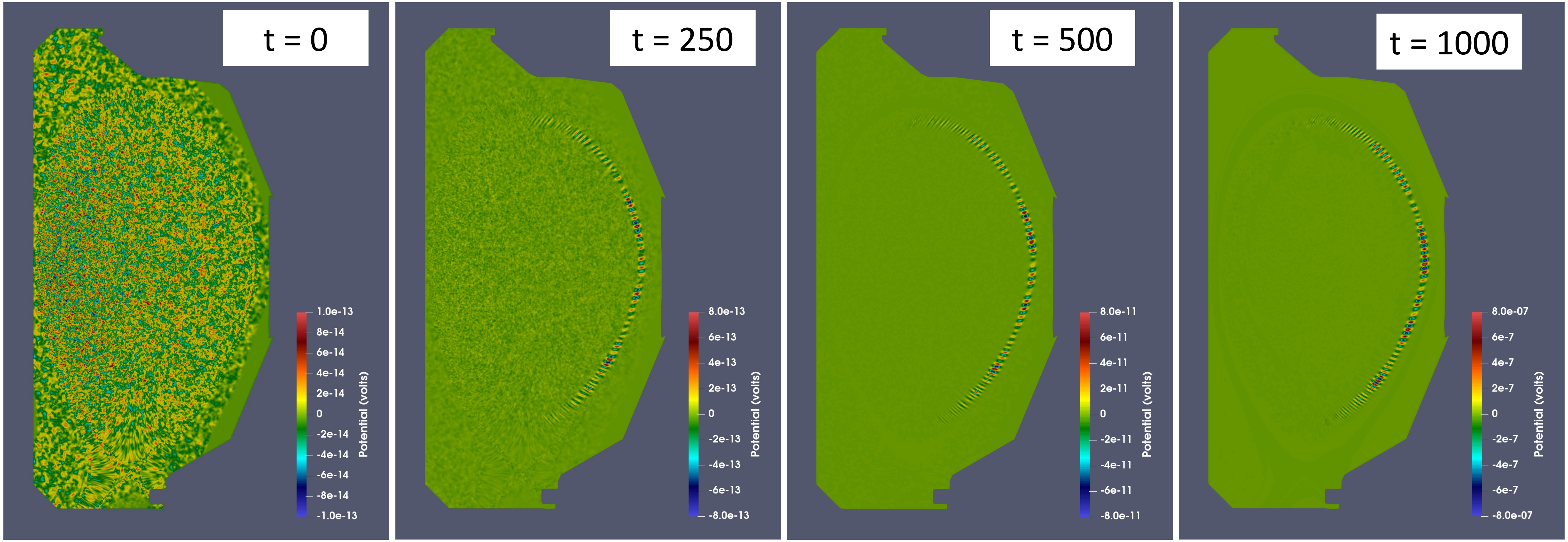


Simulation mesh, coarse mesh is shown here for visualization



Resulting turbulent electrostatic potential on one poloidal plane over time; the result show here is the mean-square-averaged potential over all mesh vertices.

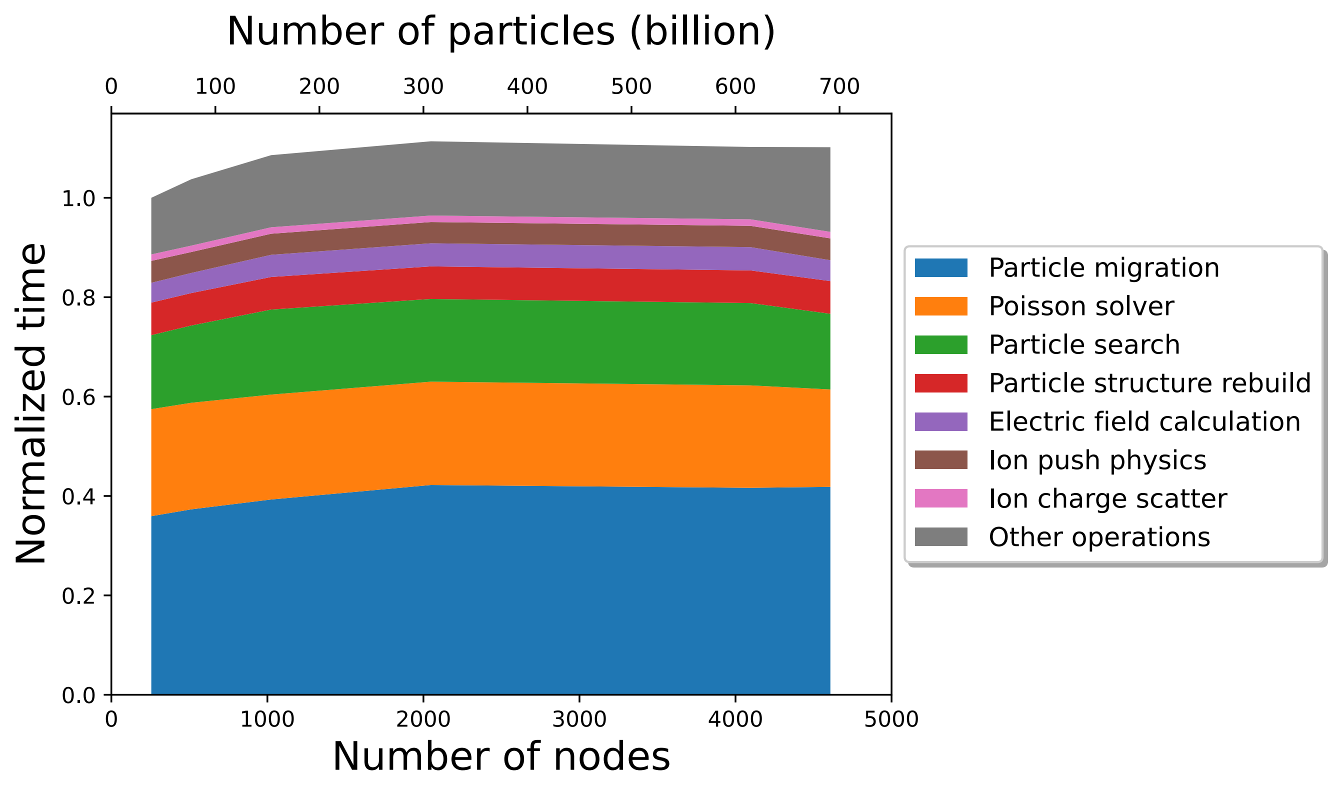
ITG with DIII-D geometry: turbulent electrostatic potential



Contour plot of turbulent electrostatic potential on one poloidal plane at different time steps

XGCm performance: weak scaling on OLCF Summit

- Summit is hosted at Oak Ridge Leadership Computing Facility (OLCF). Currently the 5th fastest computer in the world;
- Weak scaling: each GPU does same amount of work, evaluate performance as number of GPUs increase (increased problem size);
- Used 256 to 4,608 Summit computing nodes (1,536 to 27,648 GPUs);
- Up to full Summit's computing power;
- Straight line means perfect scaling.



Total simulation time cost, and time cost of major components of the code. Problem size scales with the computing nodes used[1, 2].

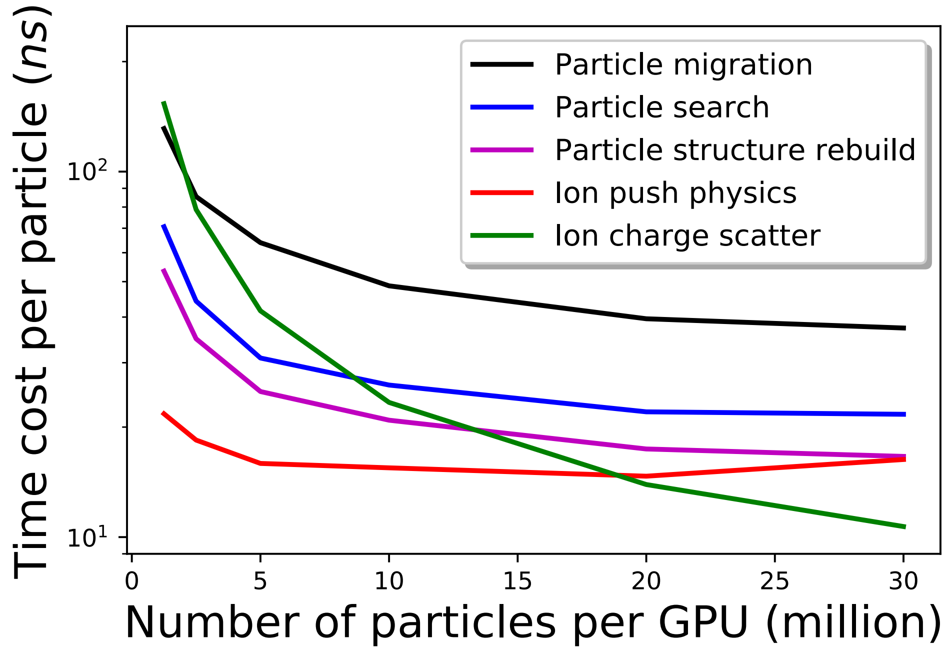
[1] C. Zhang, G. Diamond, C. W. Smith, M. S. Shephard, in review, *Computer Physics Communications*, 2023.
[2] C. Zhang, G. Diamond, C. W. Smith, M. S. Shephard, *64th Annual Meeting of the APS Division of Plasma Physics* , October 17-21, 2022, Spokane, WA.

XGCm performance: particle operations GPU kernels time cost

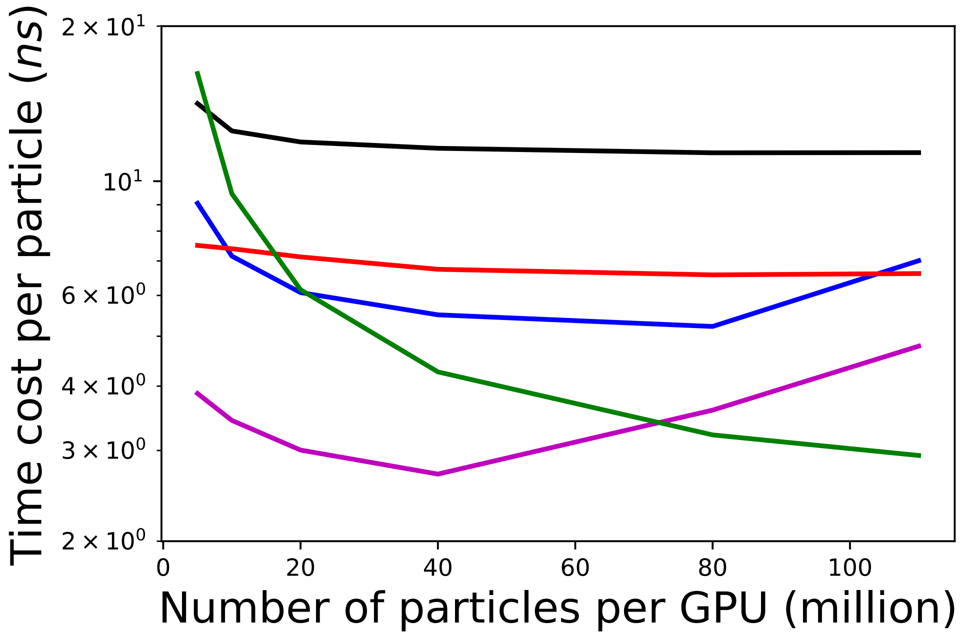
Cyclone base case with circular geometry

- Same case as previous weak scaling study;
- 8 poloidal planes, or 8 GPUs used.

Most of these GPU kernels are memory bound



590,143 mesh elements, 8 poloidal planes, 8 GPUs

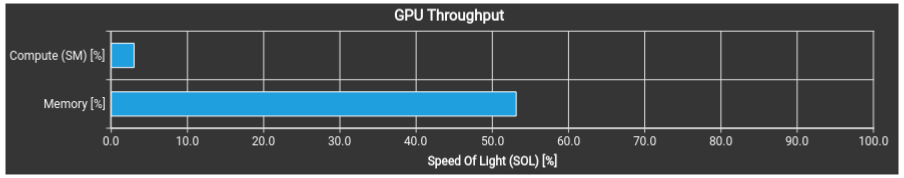


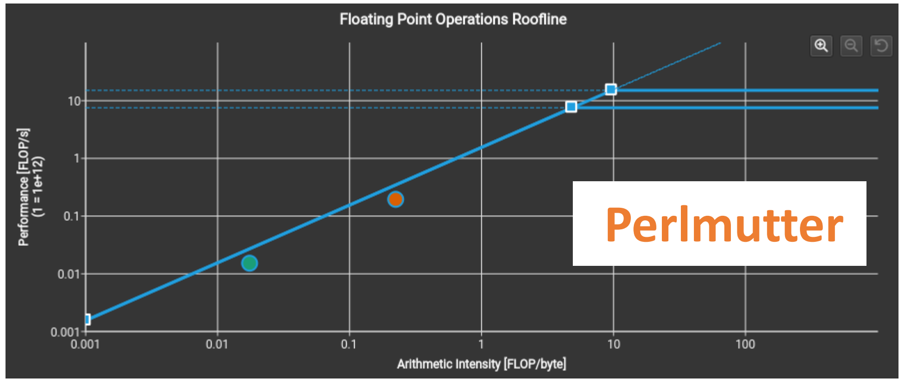
590,143 mesh elements, 8 poloidal planes, 8 GPUs

- ❑ **Need 20-30 million particles/GPU to be efficient on Summit (Nvidia V100) ;**
- ❑ **50-80 million particles/GPU on Perlmutter (Nvidia A100).**

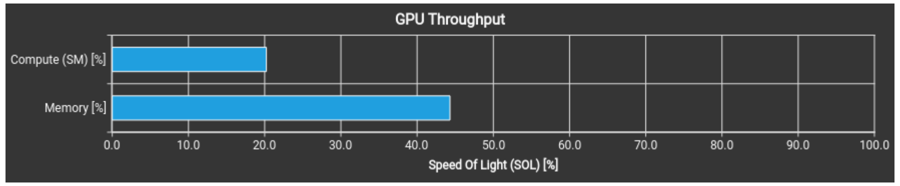
XGCm performance: simple Nvidia Nsight Compute analysis

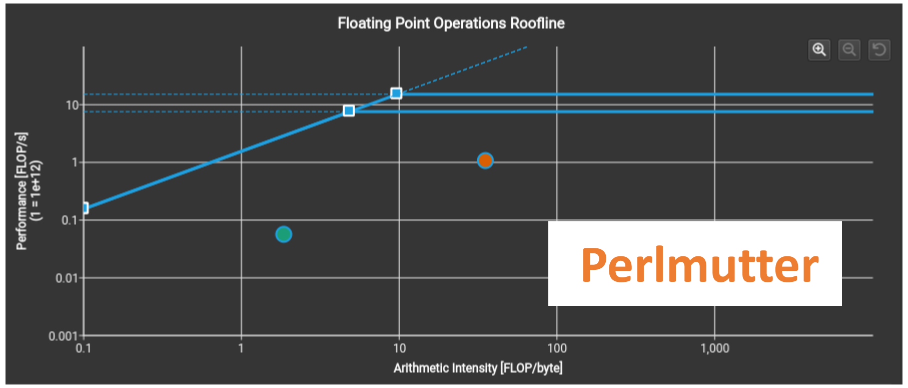
Charge scatter

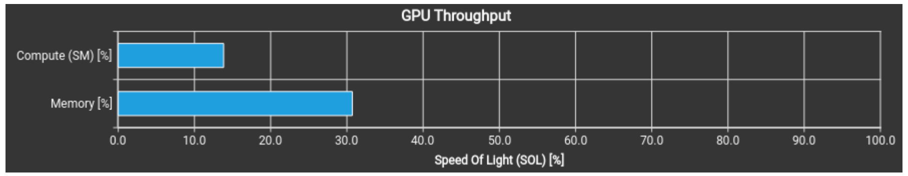


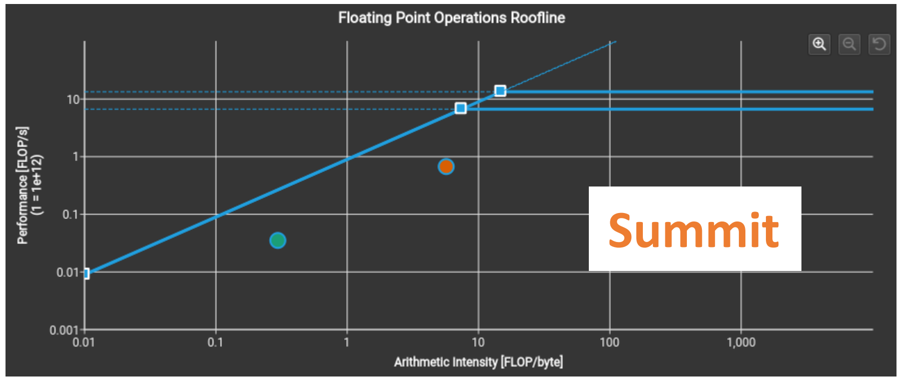


Ion push







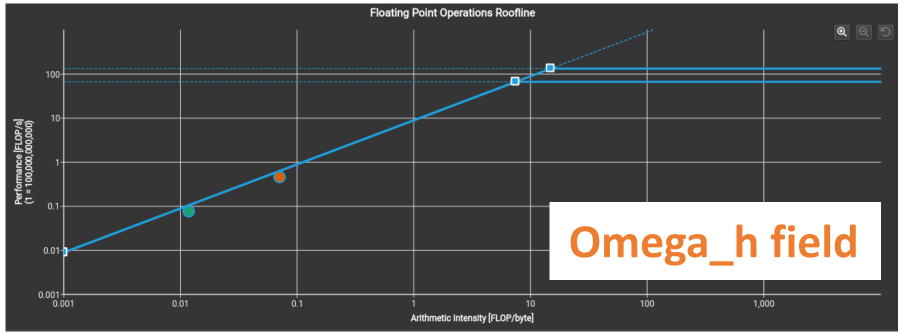


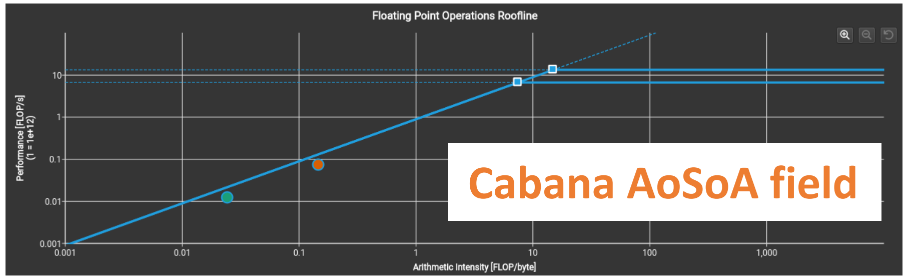
- ❑ Charge scatter kernel is heavily memory bound;
- ❑ Ion push kernel is less memory bound.

XGCm performance: simple Nvidia Nsight Compute analysis

Gyroaveraged electric field calculation

- Mesh field operation, from field A to field B;
- Operating on each **mesh vertex** of field A;
- Calculating field B defined on each mesh vertex;
- Field A and B have different components per mesh vertex.





- ❑ Simple test using Cabana's array-of-structs-of-arrays data structure, **AoSoA;**
- ❑ Roughly a 36% reduction in kernel time.

```
using DataTypes = Cabana::MemberTypes<double[NRP1*NCOMPS],double[NRP1*NCOMPS]>;
const int VectorLength = 32;
using MemorySpace = Kokkos::CudaSpace;
using ExecutionSpace = Kokkos::Cuda;
using DeviceType = Kokkos::Device<ExecutionSpace, MemorySpace>;
Cabana::AoSoA<DataTypes, DeviceType, VectorLength> aosoa("my_aosoa", mesh->nverts());
auto eff_major_slice = Cabana::slice<0>(aosoa);
auto eff_minor_slice = Cabana::slice<1>(aosoa);
```

Future work

- ❑ Mesh operations
  - ➢ Explore different unstructured mesh field storage on GPU;
  - ➢ meshFields library being developed at RPI: https://github.com/SCOREC/meshFields.
    - ▪ Use the Cabana AoSoA data structure;
    - ▪ Better data locality and performance.
- ❑ Particle operations
  - ➢ Particle push: ion and electron have dramatically different mass and hence speed;
  - ➢ Explore the performance of different particle data structures on different species.
    - ▪ Sell-C-sigma, Cabana AoSoA, DPS.
- ❑ Better use of PETSc
  - ➢ Integrate latest PETSc release with XGCm (currently using v3.16.6);
  - ➢ Currently only used Cuda, explore using Kokkos, HIP, and SYCL with different hardwares;
  - ➢ Best practice using PETSc on different GPUs vs on CPU.

Acknowledgement

- This research is supported by the DOE SciDAC program through grant DE-SC0018275 (Unstructured Mesh Technologies for Fusion Simulation Codes).
- This work is carried out in collaboration with:
  - FASTMath SciDAC Institute;
  - High-Fidelity Boundary Plasma Simulation SciDAC Partnership;
  - COPA: ECP Co-Design Center for Particle Applications.