Performance Enhancements of XGC

E. D'Azevedo[1], A, Scheinberg[2], M. Shephard[3], P. Worley[4], S. Sreepathi[1], B. MacKie-Mason[5], T. Willians[5], and the SciDAC HBPS XGC Team

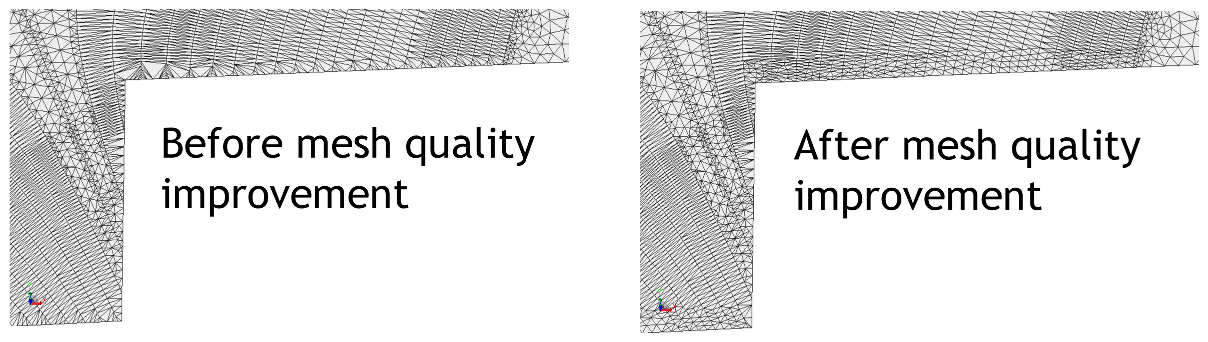
1. Oak Ridge National Laboratory , 2. Princeton Plasma Physics Laboratory, 3. Rensselaer Polytechnic Institute, 4. PHWorley Consulting, 5. Argonne National Laboratory

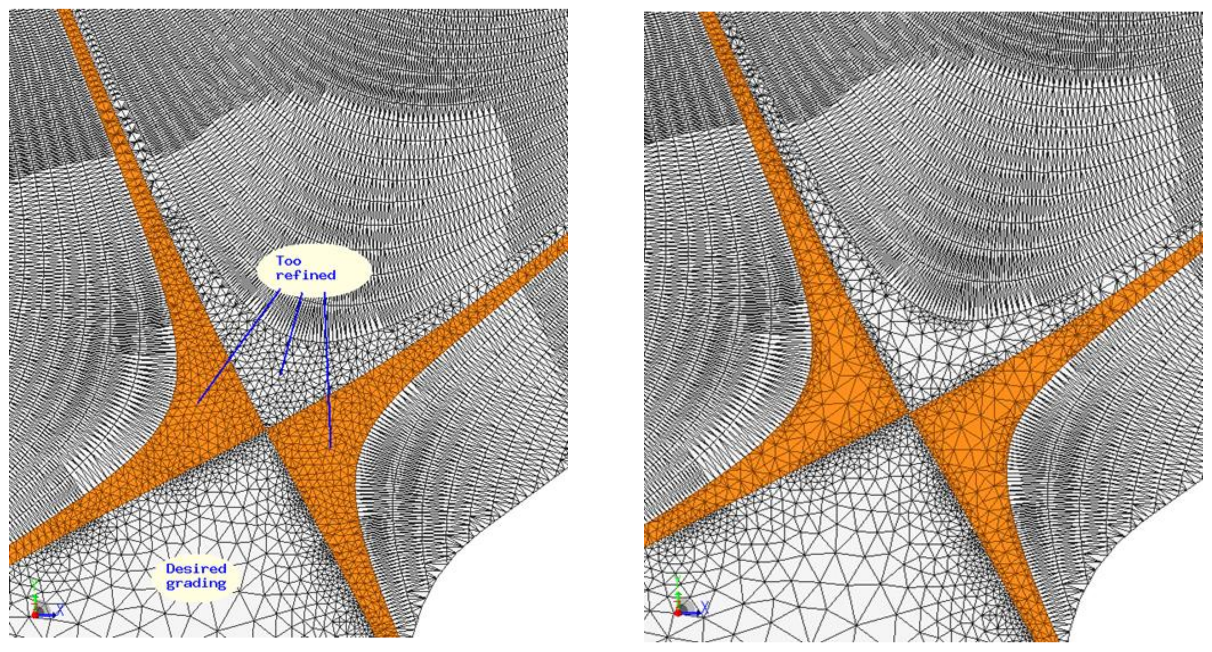
Funding is from DOE ASCR and FES Offices

HBPS High-fidelity Boundary Plasma Simulation

XGC Meshing

- Improved mesh quality in areas where flux curves interact with reactor wall
- Improved matched mesh gradation at x-point
- Reordering of mesh data for better memory access during XGC simulations



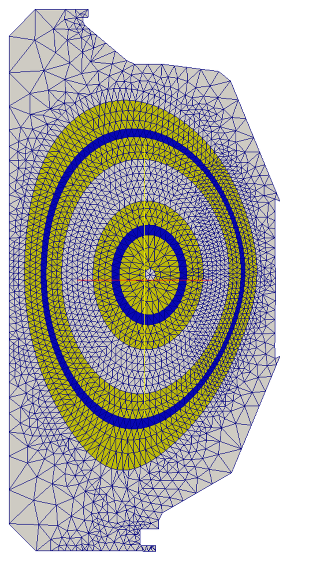


Improved mesh gradation at X-point

XGC based on Parallel Unstructured Mesh PIC (PUMIpic)

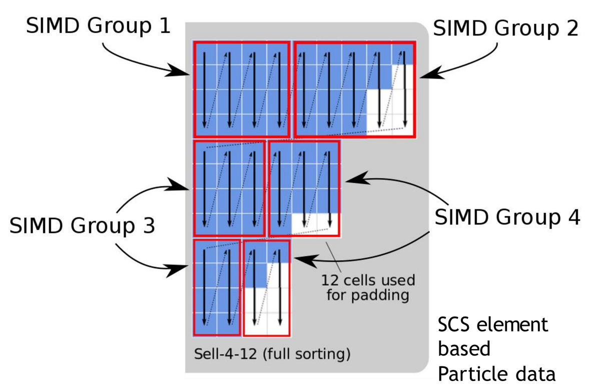
PUMIpic - Components to support PIC operations on distributed unstructured meshes (2D and 3D)

- Mesh centric – no independent particle structure
- Distributed mesh with overlaps (PICparts)
- Particle migration and load balancing between pushes
- Adjacency-based particle containment determination
- Focused on structures for execution on GPUs
  - Omega GPU ready mesh topology being integrated
  - Particles stored by element in new SCS data structure
  - Test shows on-par performance using less memory



Two PICparts

	no sorting	full sorting
ptcls (Ki)	time (s)	time (s)
128	2.298661	3.642041
256	2.895464	3.415048
512	3.79263	3.851178
1024	4.972283	4.090044
2048	7.089673	4.389198
4096	11.578984	4.799475



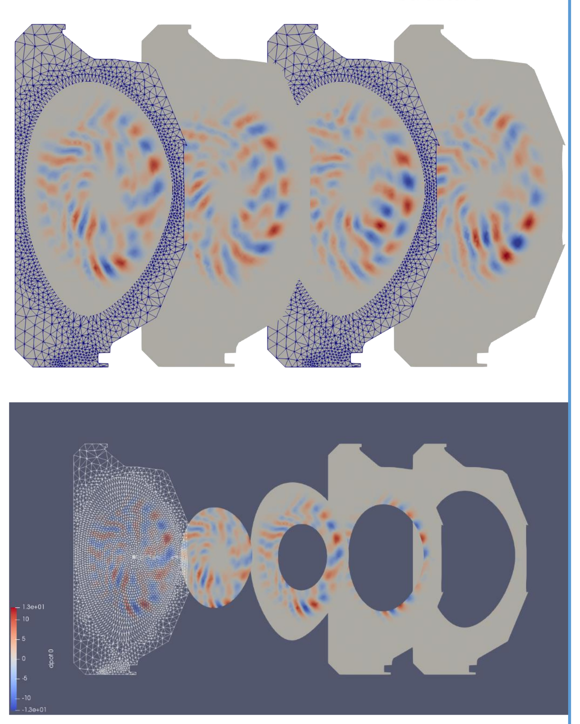
SCS element based Particle data structure

Implementing XGC physics and Numerics with PUMPIpic:

- Since all core data structures are changed code, code being rewritten in C++

Status of implementation:

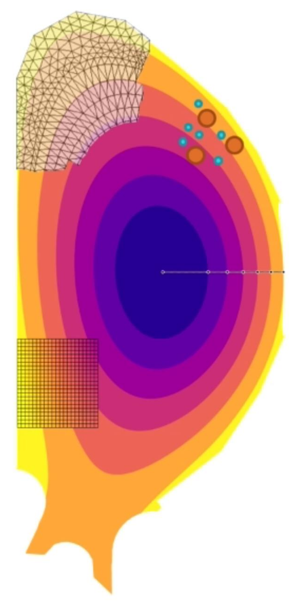
- Based on original PUMI structures – new GPU focused structures will be integrated when complete
- Core mesh/particle interaction operations in place
- Mesh solve in place
- Ion and electron push (including subcycling) implemented
- Initial δf simulations executed
- Performance evaluation and improvement underway
- Initial push results show 25% improvement on many core system
- Other steps slower due to need to modify mesh copies (underway)



Snapshot of electrostatic potential fluctuation (a) at toroidal angle $\zeta=0,\pi/2,\pi,3\pi/2$ from left to right and (b) in local domain of each group at $\zeta=0$

XGC on Summit

- XGC Gyrokinetic particle-in-cell (PIC) code is also part of ECP-WDM (whole device model) and ECP-CoPA (particle app co-design)
- XGC is part of Early Science Programs on Summit, Aurora and Perlmutter
- XGC uses an unstructured grid in poloidal plane, each MPI rank gets particles from a section of poloidal plane
- Main computational kernel is electron push
- Utilizes Kokkos via Cabana of CoPA

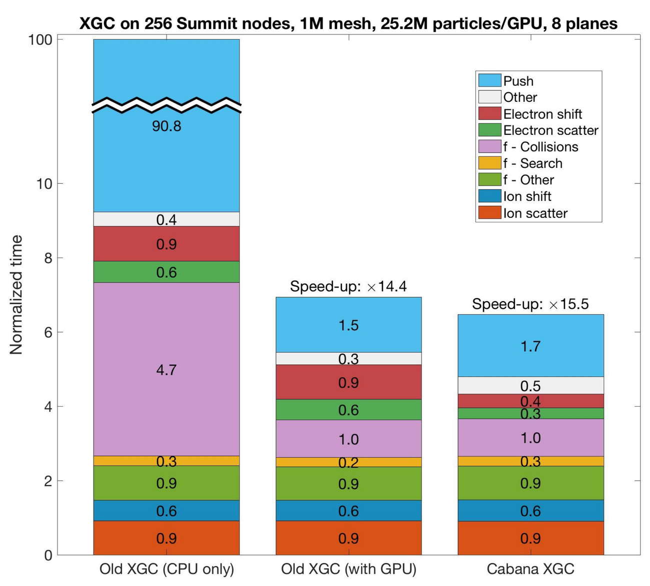


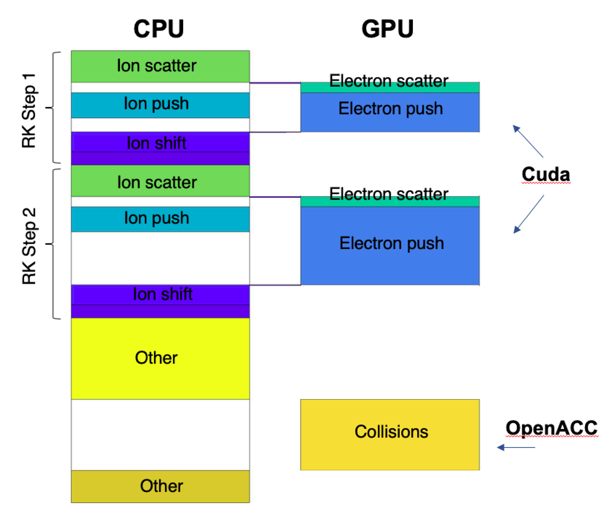
XGC_core/pushe.F90:

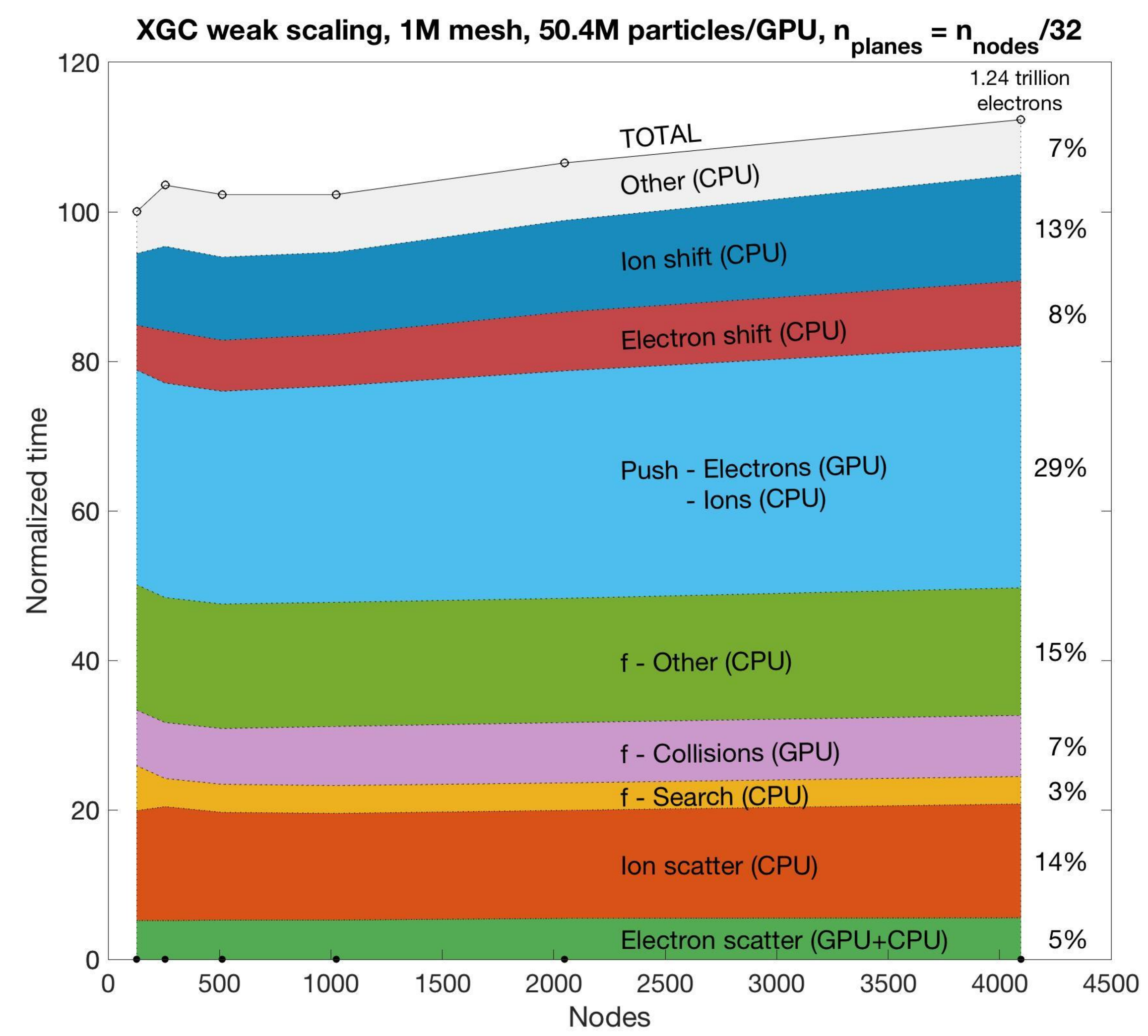
```
subroutine pushe
  call sort_particles              ! Sort particles by grid cell
  do iptl=1, n_particles           ! Loop over particles
    do ic=1, n_cycles              ! Subcycle electrons
      do irk=1, n_runge_kutta      ! RK4 loop
        call search                  ! Determine which grid cell particle inhabits
        call gather_field          ! Interpolate field at particle location
        call calculate_dx          ! Solve physics: dx/dt = f(E,…)
        call advance_particles     ! Update particle position and velocity
      end do
    end do
  end do
end subroutine pushe
```

Good Weak Scaling to Full Summit

- On 256 nodes of Summit, GPU version has **15X speedup over CPU only**
- Good weak scaling up to full Summit using **1.24 trillion electrons on GPU and 1.24 trillion ions on CPU**







Details on Cabana Version

- XGC in Fortran, Cabana and Kokkos in C++
- Allocate particle storage in Cabana and use macros for generating Fortran interface enables easy porting of new kernels
- Single code for CPU and GPU
- Electron push kernel in CUDA Fortran (C++ version under development)

```
void main()
{
  Cabana::initialize();

  // Create instance of array of structure of arrays
  ParticleList particles( num_particle );

  // Create "range policy"
  Cabana::RangePolicy<ParticleList::array_size,ExecutionSpace> range_policy( 0, particles.numSoA() );

  // Main time loop
  for (int i=1; i<=n_steps; i++){
    ... // Deposition, field solver, etc.
    Cabana::parallel_for( range_policy_vec, push_electrons, Cabana::IndexParallelTag() );
  }

  Cabana::finalize();
}

// Electron push subroutine is a lambda function
auto push_electrons = KOKKOS_LAMBDA( const int idx )
{
  ...
};
```

```
num_vecs=num_particle/vector_length
... ! Platform-specific directives etc.
do i_vec=1,num_vecs
  call push_electrons(all_particles(i_vec),i_vec))
end do
```

```
! Macro generating Fortran INTERFACE
PARTICLE_OP_F(pushe)

subroutine pushe_f(particle_vec, i_vec) BIND(C,name='pushe_f')
  USE, INTRINSIC :: ISO_C_BINDING
  type(ptl_type) :: particle_vec
  integer(C_INT), value :: i_vec
  do i=1, vector_length
    ... ! Vectorizable loop that advances particle positions
  end do
end subroutine
```

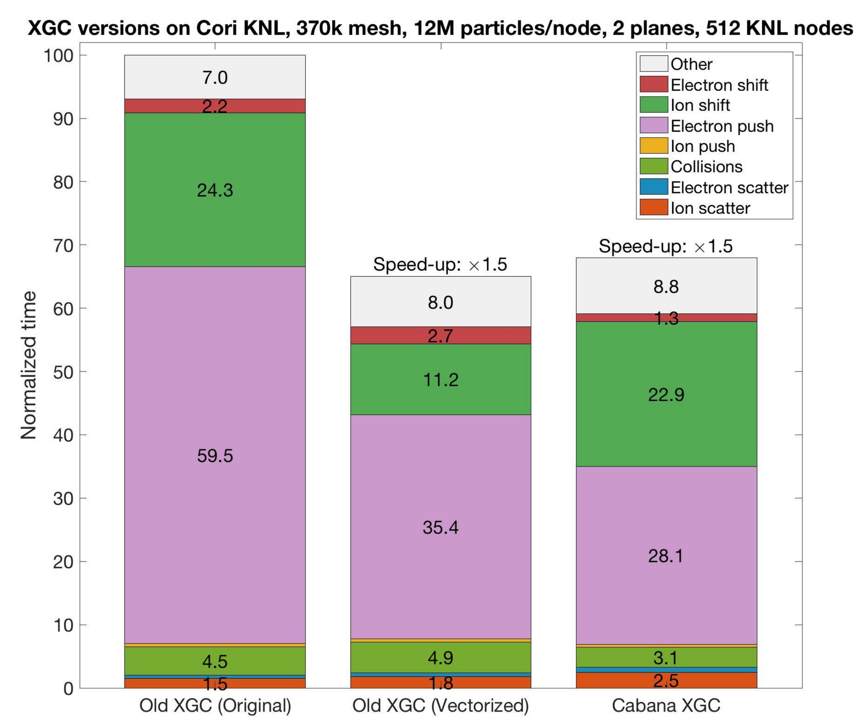
Must cast Cabana array into predefined Fortran type for use in Fortran kernels using ISO_C_BINDING

```
// Create Cabana structure type
using ParticleDataTypes =
      Cabana::MemberDataTypes< double[6], double[3], int >;

using ArrayLayout =
      Cabana::InnerArrayLayout<vector_length,Cabana::LayoutLeft>;
using ParticleList =
      Cabana::AoSoA<ParticleDataTypes,MemorySpace,ArrayLayout>;

// Create instance of array of structure of arrays
ParticleList particles( num_particle );


// Create analogous C structure of arrays
struct local_particle_struct {
    double ph[6][vector_length];
    double ct[3][vector_length];
    int gid[vector_length];
};

// Define array of pointers to particle structures
auto* p_loc = (local_particle_struct*)(particles.ptr());
```

```
module ptl_module
  use, intrinsic :: ISO_C_BINDING
  type, BIND(C) :: ptl_type
     real (C_DOUBLE) :: ph(vector_length,6)
     real (C_DOUBLE) :: ct(vector_length,3)
     integer (C_INT) :: gid(vector_length)
  end type ptl_type
end module
```

Performance on KNL

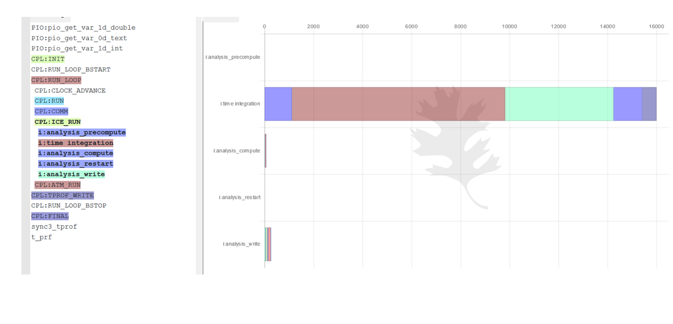
- Cabana version of XGC has been ported to Cori KNL
- Roofline analysis of vectorized version of XGC shows in-lining and re-factoring useful in optimizing use of wide-vector registers. However, vector dependences and data type conversions limiting peak performance



Performance Analytics for Computational Experiments for XGC

- Central hub of performance data, already used in Climate application
- Interactively deep-dive and track performance benchmark
- Facilitate performance analysis:
  - Load balancing
  - Identification of bottlenecks
  - Inform targeted optimization efforts

https://pace.ornl.gov



Tree and Flame Graphs