XGC with Kokkos/Cabana: Plasma Physics on Summit and Beyond

A. Scheinberg\textsuperscript{1,4}, S. Ethier\textsuperscript{1}, G. Chen\textsuperscript{2}, S. Slattery\textsuperscript{3}, R. Bird\textsuperscript{2}, E. D’Azevedo\textsuperscript{3}, S.-H. Ku\textsuperscript{1}, CS Chang\textsuperscript{1}

In collaboration with ECP-CoPA

\textit{P3HPC 2020, Sept. 2}

\textsuperscript{1}Princeton Plasma Physics Laboratory
\textsuperscript{2}Las Alamos National Laboratory
\textsuperscript{3}Oak Ridge National Laboratory
\textsuperscript{4}Jubilee Development
## WDMApp Requires Exascale Computers and Beyond

<table>
<thead>
<tr>
<th>Giga-flops</th>
<th>Tera-flops</th>
<th>Peta-flops</th>
<th>Exa-flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-D electrostatic ion physics in simplified circular cylindrical geometry</td>
<td>Core: 5D ion-scale electromagnetic physics in torus</td>
<td>Core: 5D ion scale Maxwellian ion + electron electromagnetic</td>
<td>Core-edge coupled 5D electromagnetic study of whole-device ITER, including ion-scale turbulence + local electron-scale turbulence, profile evolution, large-scale instability, plasma-material interaction, rf heating, and energetic particles</td>
</tr>
<tr>
<td>Edge: ion+neutral electrostatic physics in torus</td>
<td>Edge: non-Maxwellian plasma, electrostatic physics</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Beyond**

A WDMApp that includes necessary engineering reactor components, and applicable to leading alternate concepts (including stellarators); and possibly 6D whole device modeling.
XGC outline

• Gyrokinetic (i.e. 5D) particle-in-cell code on an unstructured grid
XGC outline

- Gyrokinetic (i.e., 5D) particle-in-cell code on an unstructured grid.
Why adopt Cabana and Kokkos?

• Portability!

• Let Kokkos and Cabana handle **data management** and **kernel execution** for easy portability between architectures

• Reduce compiler dependencies (e.g. only PGI on Summit)

• Provide an easy/flexible framework for porting more kernels to GPU

• Avoid code duplication
  – 3 previous versions: original, vectorized, Cuda
Why adopt Cabana and Kokkos?

• Portability!

• Let Kokkos and Cabana handle **data management** and **kernel execution** for easy portability between architectures

• Reduce compiler dependencies (e.g. only PGI on Summit)

• Provide an easy/flexible framework for porting more kernels to GPU

• Avoid code duplication
  – 3 previous versions: original, vectorized, Cuda, **Cabana**?
Old implementation of XGC

Old setup:
- All Fortran/Cuda Fortran

How does a Fortran code adopt a C++ programming model?
A Kokkos implementation of XGC

New setup:
- Keep Fortran main and kernels
- C++ interface
  - “Light touch:” Localized modification
  - Gradual implementation
- Unified, optimized code base
Data layout with Cabana

• Cabana (ECP-CoPA): a library for particle-based applications
  – Built on Kokkos
  – Provides AoSoA (array of structures of arrays) for versatile layout

```cpp
// Define Cabana structure type
using ParticleDataTypes = Cabana::MemberDataTypes<
  double[6],
  double[3],
  int >;
```
Executing the Kokkos parallel_for

- Kernel is called in a Kokkos parallel_for

```cpp
Kokkos::RangePolicy<ParticleList::array_size,ExecutionSpace> range_policy( 0, n_items ); // n_ptl on GPU, n_structs on CPU

// Execute parallel_for
Kokkos::parallel_for("my_operation", range_policy_vec, KOKKOS_LAMBDA( const int idx )
{
    push_f(p_loc+idx, idx);
});
```

- Must cast Cabana array into Fortran type for use in Fortran kernels

```fortran
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
```

- Inner loops for vectorization on CPU

```fortran
subroutine push_f(particle_vec, i_vec) BIND(C,name='push_f')
    USE, INTRINSIC :: ISO_C_BINDING
    type(ptl_type) :: particle_vec
    integer(C_INT), value :: i_vec

    do i=1, simd_size ! 32 on CPU, 1 on GPU
        ... ! Vectorizable loop that advances particle positions
    end do

end subroutine
```
Timing on Summit (256 nodes)

- Overall speed-up: 15x CPU only
- CPU-GPU communication costs low
  - Actual electron transfer time shown
  - Favors simple approach to communication
Summit performance comparison and scaling

XGC on 256 Summit nodes, 1M mesh, 25.2M particles/GPU, 8 planes

Old CPU version

Old specialized Cuda version

Cabana version

Speed-up: $\times 14.4$

Speed-up: $\times 15.5$

XGC weak scaling, 1M mesh, 50.4M particles/GPU, $n_{\text{planes}} = n_{\text{nodes}}/32$
Cori KNL performance comparison and scaling

XGC versions on Cori KNL, 370k mesh, 12M particles/node, 2 planes, 512 KNL nodes

Old CPU version

<table>
<thead>
<tr>
<th>Category</th>
<th>Old CPU version</th>
<th>Old specialized CPU version</th>
<th>Cabana version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized time</td>
<td>59.5</td>
<td>35.4</td>
<td>28.0</td>
</tr>
<tr>
<td>Speed-up:</td>
<td>×1.5</td>
<td>×1.8</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>4.5</td>
<td>4.9</td>
<td>4.7</td>
</tr>
<tr>
<td>Electron shift</td>
<td>1.5</td>
<td>1.9</td>
<td>2.0</td>
</tr>
<tr>
<td>Ion shift</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electron push</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ion push</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collisions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electron scatter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ion scatter</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XGC weak scaling, 370k mesh, 12M particles/node, \( n_{\text{planes}} = \frac{n_{\text{nodes}}}{256} \)

- TOTAL
- Other
- Electron shift
- Ion shift
- Electron push
- Ion push
- Collisions
- Electron scatter
- Ion scatter

- 50 billion electrons
- 17%
- 5%
- 12%
- 48%
- 9%
- 2%
- 6%
Transition to C++

- Diverse architectures coming up in the near future
  - Challenges lie ahead for portability

<table>
<thead>
<tr>
<th>Supercomputer</th>
<th>Year</th>
<th>Petaflops</th>
<th>Architecture</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summit</td>
<td>2019</td>
<td>200</td>
<td>Nvidia GPUs</td>
<td>Cuda</td>
</tr>
<tr>
<td>Perlmutter</td>
<td>2020</td>
<td>100</td>
<td>Nvidia GPUs</td>
<td>Cuda</td>
</tr>
<tr>
<td>Aurora</td>
<td>2021 (?)</td>
<td>1,000</td>
<td>Intel GPUs</td>
<td>SYCL</td>
</tr>
<tr>
<td>Frontier</td>
<td>2021</td>
<td>1,500</td>
<td>AMD GPUs</td>
<td>HIP</td>
</tr>
<tr>
<td>Fugaku</td>
<td>2021</td>
<td>1,000</td>
<td>ARM</td>
<td>Fortran/C++</td>
</tr>
</tbody>
</table>

- Support generally better for C++ than Fortran
- Easier use of Kokkos/Cabana if code is in C++
The Cabana Fortran implementation of XGC

The "Cabana Fortran" implementation

- Keep Fortran main and kernels
- C++ interface
  - "Light touch:" Localized modification
  - Gradual implementation
- Unified, optimized code base
- Downsides:
  - Inflexible macros
  - No HIP/SYCL support
  - Tedious data transfer
Kokkos C++ Implementation of XGC

Current setup:

• Transition to C++ continues
  – Main loop in C++ for easier memory management

• Integrated Kokkos/Cabana

• Arrays (field etc.) passed from Fortran, copied to Kokkos views

• No explicit cuda or OpenMP

• Ready for any architectures with Kokkos and OpenACC
  – In theory
Converting the collision kernel to Kokkos

Motivation:

- Pitfalls of multiple programming models (Kokkos and OpenACC)
  - Memory management
  - Compiler compatibility
  - More opportunities for something to go wrong
- Converting to C++ anyway
Converting the collision kernel to Kokkos

- Problem: Collisions computed separately for each mesh node
  - ~5,000 mesh nodes per GPU
  - ~20 Kokkos kernels each
  - Kernels loop over ~1,000 elements -> GPUs underutilized
  - Some calculations still on CPU (harder to port) -> More GPU idle time
Converting the collision kernel to Kokkos

- **Approach: Multiple streams**
  - Already done in our OpenACC implementation
  - Kokkos also supports Cuda streams
  - OpenMP parallel region, each OpenMP thread gets its own Cuda stream

- **Result:**
  - GPU usage much higher
  - 25% speed-up from OpenACC Fortran version
  - Still room for improvement (2-4x)?

- **Downside: Possible portability challenges**
  - Will multiple streams be a viable option for various Kokkos back-ends and architectures?
Summary

• XGC with Kokkos/Cabana is performing well on Summit and Cori KNL

• All major kernels offloaded to GPU with Kokkos
  – Electron push, collisions; also charge deposition, sorting

• More compiler flexibility (no longer tied to PGI on Summit)

Future challenges

• Moving more XGC kernels to Cabana framework
  – More optimization possible

• GPU-GPU communication
  – Potentially rely on Cabana for this

• Ensuring diverging developments can benefit
  – ECP-WDM projects (coupling with GENE, GEM, HPIC for whole-device modeling) and other science goals are on different branches