Kokkos and Fortran in the Exascale Computing Project plasma physics code XGC

A. Scheinberg,1,∗ G. Chen,2 S. Ethier,1 S. Slattery,3 R. Bird,2 P. Worley, C.S. Chang1 1Princeton Plasma Physics Laboratory, 2Los Alamos National Laboratory, 3Oak Ridge National Laboratory

Introduction
Numerical plasma physics models such as the particle-in-cell XGC code are important tools to understand phenomena encountered in experimental fusion devices. Here we adapt XGC to use Kokkos, a programming model for portable supercomputer performance, and Cabana, a recent library built on Kokkos as part of the ECP-Copa project containing kernels and operations typically necessary for particle-based scientific codes. We summarize the use by XGC of part of the ECP-CoPA project containing kernels and operations typically necessary for particle-based scientific codes. We show that this approach can provide a single, portable code base that performs well on both GPUs and multicore machines.

Model: XGC
XGC is a particle-in-cell plasma physics code which uses a gyrokinetic model to simulate plasma turbulence in the edge and core of magnetically confined fusion devices. Markers denoting the ion and electron distribution functions are distributed in configuration space. The electric field, stored on an unstructured grid, is evaluated at the marker positions, which are updated (“pushed”) accordingly. The new charge distribution is then mapped back to the grid, which are updated (“pushed”) accordingly. The new electron distribution functions are distributed in configuration space.

Kernel execution
The Fortran main program calls a Kokkos C++ function, which in turn calls a Fortran kernel using ISO C Binding:

```fortran
void electron_push_dispatch(int N) // called by Fortran main
auto push_lambda = KOKKOS_LAMBDA( const int idx )
electron_push(idx); // call to Fortran kernel
Kokkos::parallel_for( range_policy, push_lambda )
```

This interface and others were shortened to convenient macros:

```cpp
KOKKOS_OP(electron_push_dispatch, electron_push)
```

Data layout and transfer
The particles use Flexible Array of Structures of Arrays data layout in Cabana. If on GPU, the parallel_for loops over particles; if on CPU, it loops over vectors:

```fortran
N = USE_GPU ? N_PTF + N_PTF / VEC_LEN :
Kokkos::RangePolicy(execution_space) range_policy( 0, N);
```

Asynchronous execution
Idle time on both the host and device must be minimized for efficient resource use. Increased modularity, in our case separation of ion tasks from electron tasks, can remove unnecessary sync points and enable asynchronous CPU and GPU operation which reduces idle time. Communication can also be done asynchronously, but this was not prioritized since, due to fast data transfer on Summit, host-device communication is only ~1% of total runtime.

Conclusion
• Using the Kokkos/Cabana framework, we produced a production-ready code that matches performance of an already highly optimized HPC code.
• Due to Cabana’s ease of use, other parts of the code that were previously un-optimized were rapidly vectorized and ported to GPU.
• The framework will allow other scientific programmers to focus more on science and less on optimization, thus its flexibility leaves plenty of opportunities for optimizations within and between kernels.
• Our macros and simple examples of the Fortran Cabana approach can be found at github.com/ECP-copa/Cabana; however, we suggest converting to C++ for maximum portability on upcoming architectures.
• In either language, attention to modularization, data layout, and hierarchical parallelism are key.