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

Aaron Scheinberg, Stephane Ethier, Choong-Seock Chang
ascheinb@pppl.gov
Princeton Plasma Physics Laboratory
Princeton, New Jersey

Guangye Chen, Robert Bird
Los Alamos National Laboratory
Los Alamos, New Mexico

Stuart Slattery
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Pat Worley
PHWorley Consulting

ABSTRACT

Numerical plasma physics models such as the particle-in-cell XGC code are important tools to understand phenomena encountered in experimental fusion devices. Adequately resolved simulations are computationally expensive, so optimization is essential. To address the need for consistent high performance by cutting-edge scientific software applications, frameworks such as Kokkos have been developed to enable portability as advances in supercomputing require hardware-specific coding implementation for best performance. Cabana, a recent extension to Kokkos developed with the ECP-CoPA project, is a library of common kernels and operations typically necessary for particle-based scientific codes. As such, the Kokkos/Cabana framework enables intuitive construction of particle-based codes, while maintaining portability between architectures. Here, we summarize the adoption by XGC of the execution and data layout patterns offered by this framework. We demonstrate a method for Fortran codes to adopt Kokkos and show that this approach can provide a single, portable code base that performs well on both GPUs and multicore machines.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; • **Applied computing** → **Physics**.

KEYWORDS

plasma physics, high performance computing, parallel computing

ACM Reference Format:
Aaron Scheinberg, Stephane Ethier, Choong-Seock Chang, Guangye Chen, Robert Bird, Stuart Slattery, and Pat Worley. 2019. Kokkos and Fortran in the Exascale Computing Project plasma physics code XGC. In *Proceedings of Supercomputing '19*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Supercomputing '19, November 17–22, 2019, Denver, CO

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

XGC is a particle-in-cell plasma physics code written in Fortran that uses a gyrokinetic model to simulate plasma turbulence in the edge and core of magnetically confined fusion devices.[2] Markers containing information about the ion and electron particle distribution function are distributed in phase 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, where the electric field is solved for the next time step. Due to their high velocity relative to the ions, electron position must be updated on a shorter time scale than the ion positions or (in some physics applications) the electric field. The electrons are thus "sub-cycled", typically moving 60 steps per ion step (and per field solve), and comprise the bulk of computation. Different versions of this "electron push" kernel were developed to maximize performance on specific hardware: a Cuda Fortran version optimized for the previous Oak Ridge supercomputer Titan; an OpenMP version that vectorizes and performs well on CPUs; and an unvectorized OpenMP version maintained as a reference.

Programming models such as Kokkos, introduced to provide a generic interface for optimized use of heterogenous architectures,[1] are typically written in C++ due to its flexibility, object-oriented nature, and community support. To allow Fortran-based applications to take advantage of new developments in portability frameworks, we have developed a workflow to embed C++ frameworks into a Fortran code with minimal changes to the scientific application itself. While not as simple as a "plug-and-play"-type interface, our implementation allows developers of scientific Fortran applications to make basic use of Kokkos and thereby to reduce the need for hardware-specific coding.

2 PARTICLE OPERATION PROCEDURE

To make use of the Kokkos/Cabana framework in Fortran, data layout and particle operations, constructed in C++, must be available for use in the Fortran code. The key data layout for Cabana, array of structures of arrays (AoSoA), is initialized in a C++ function which can be called from the Fortran main program. The pointer to this data (or subsets of it) is then passed to Fortran kernels which interpret the pointer as a reference to a scalar or array of a Fortran user-defined type (the "structure") corresponding to the size of the

structures initialized by Cabana. If passing the address of a structure, an analogous C++ class must be defined of identical size in memory in order to correctly determine the address of a particular structure within the AoSoA.

These Fortran kernels can then be called from Kokkos functions for given memory and execution policies. Since they should be interpretable by Cuda, each kernel is preceded by a macro which expands to indicate it is a device subroutine if using Cuda, or is empty if not. C++ containing these Kokkos operations can be called from the Fortran main program. In general, whatever data the kernel may need from the main program can be passed to the C++ function and from there into the kernel. In practice, it is more expedient to pass only the range of particles that the operation should act upon into the C++ function, and pass only the pointer and the structure index into the Fortran kernel. Additional data can be accessed by the kernels by placing it in Fortran modules that are common to both the kernel and the main program. In this way, the C++ function is general for all particle operations. We created a macro containing this common subroutine. Each particle operation is then an instance of this macro, which links the C++ function called by main with the Fortran kernel to be executed.

As such, a Fortran user need only populate an include.h file denoting these function pairs (the C++ function and the Fortran kernel). The automatically generated C++ function can then be called from the Fortran main program without additional, often device-specific parallelization programming.

In XGC, we now execute particle operations with the Kokkos `parallel_for`. In the CPU version, where vectorization is key, our `parallel_for` loops over the structures of the AoSoA and passes them in. As each of these structures contains the attributes of a certain number of particles, `VEC_LEN`, a loop over the particles present in a single structure is conducted in the Fortran code. This approach leaves some parallelization user-facing; however, this is also a benefit in that users are constricted to writing algorithms that can be vectorized.

On GPU, it is desirable to loop over every particle individually so that a single warp of Cuda threads is handed consecutive particles for performant data coalescence. Since maintaining a single code base is a major goal, the loop present in the kernels for CPU vectorization remains; however, it iterates only once, on the specific particle handed it by the `parallel_for`.

2.1 Race conditions

Grid-to-particle operations involve multiple particles writing to the same memory location. These operations are typically handled differently on CPU and GPU. On CPU, array replication is typical, while on GPU Cuda atomics are the most effective approach. To address this, we add an additional dimension to the array being written to. If the execution space is CPU, then this dimension is of length `OMP_NUM_THREADS`; if not, the dimension is length 1. A macro is then used which conducts the addition operation on the sub-array corresponding to the OpenMP thread number if on CPU, or conducts a Cuda atomicAdd if on GPU. A reduction is then performed manually in the CPU version.

3 ADOPTION

With the tools described above, we began the process of shifting XGC's particle operations into the Cabana framework. As XGC is $\sim 10^5$ lines of code, this is a long-term task. Fortunately, the framework developed above allows us to move one kernel at a time. So far, the electron push kernel, the charge deposition kernel and the sections of the particle shift use Cabana. Since XGC's particles are stored in an AoS data layout, the particle layout must be converted prior to the Cabana kernels, and converted back afterwards. Once the remaining electron operations are addressed, the original particle data structure can be fully removed. A similar process will begin for ions.

The piecemeal nature of this approach allowed for easy adoption as issues could be located rapidly and validity could be verified continuously.

4 PERFORMANCE

Performance testing was done on two machines: Summit, to test our optimized code on GPU, and Cori KNL, to test performance on multicore CPU systems. Tests were run with simulation parameters and size comparable to simulations conducted for scientific production and compared against the existing versions of the code. Figure 1 compares the performance of the Cabana version against the original CPU version and the pure Cuda Fortran version on Summit. Figure 2 compares the Cabana version against the original CPU version and the vectorized version on Cori KNL. The Cabana version outperforms both. The improvements cannot be entirely attributed to this adoption, since minor algorithmic/structural changes necessarily occurred during the porting process. However, the testing demonstrates that, utilizing Kokkos/Cabana, performance need not be sacrificed for the sake of portability.

5 CONCLUSION

Using the Kokkos/Cabana framework, we have produced a production-ready code that matches performance of an already highly optimized HPC code. Furthermore, due to Cabana's ease of use, several other parts of the code that were previously un-optimized were rapidly vectorized on CPU and ported to GPU. The framework will allow other scientific programmers to focus more on science and less on optimization, though its flexibility leaves plenty of opportunities for algorithmic optimizations at the kernel level as well as at a high level, e.g. choosing what operations happen on GPU vs CPU.

ACKNOWLEDGMENTS

This work is funded by the Exascale Computing Project (ECP) Co-Design Center for Particle Applications (CoPA) project and was conducted at the Princeton Plasma Physics Laboratory, Los Alamos National Laboratory, and Oak Ridge National Laboratory.

REFERENCES

[1] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel and Distrib. Comput.* 74 (2014), 3202–3216. Issue 12. https://doi.org/10.1016/j.jpdc.2014.07.003

[2] 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. 2018. A fast low-to-high confinement mode bifurcation dynamics in the boundary-plasma

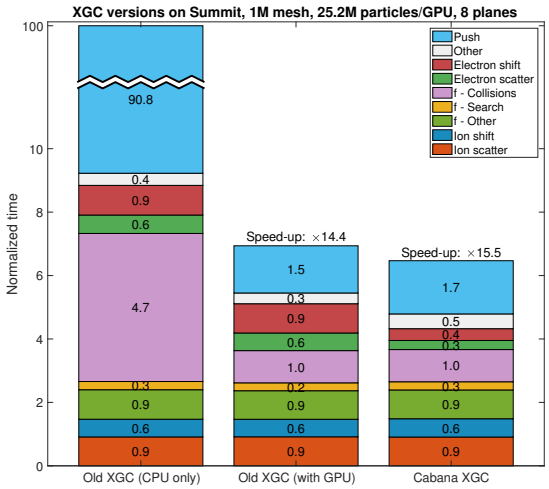

Figure 1: Performance comparison on Summit. The Cabana version of the electron push kernel is ∼10% slower than the optimized Cuda Fortran version, but the relative ease of GPU offloading with the new framework enabled rapid porting of other kernels, leading to a net acceleration. The collision kernel is implemented with OpenACC and was not altered here.

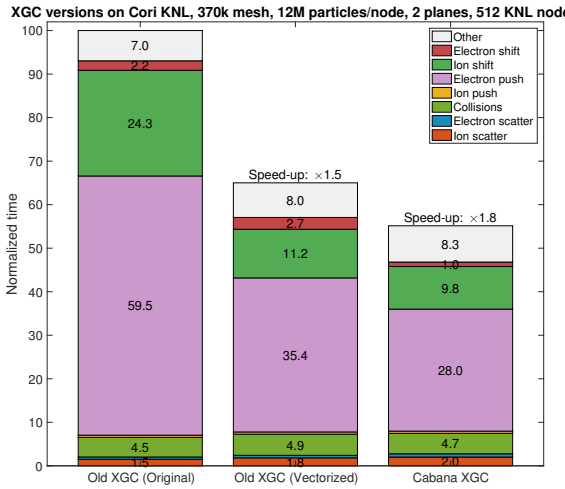

Figure 2: Performance comparison on Cori's KNL partition. The Cabana version of the electron push is substantially better-vectorized.

gyrokinetic code XGC1. *Physics of Plasmas* 25 (2018), 056107. https://doi.org/10.1063/1.5020792