Linking a Next-Gen Remap Library into a Long-Lived Production Code

Charles R. Ferenbaugh
Brendan K. Krueger
{cferenba,bkkrueger}@lanl.gov
Los Alamos National Laboratory

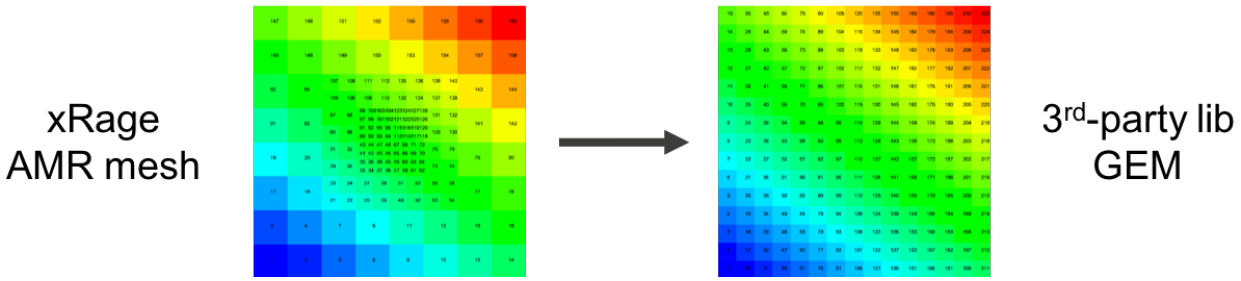

Figure 1: Remap from AMR mesh to GEM mesh.

ABSTRACT

LANL's long-lived production application xRage contains a remapper capability that maps mesh fields from its native AMR mesh to the GEM mesh format used by some third-party libraries. The current remapper was implemented in a short timeframe and is challenging to maintain. Meanwhile, our next-generation code project has developed a modern remapping library Portage, and the xRage team wanted to link in Portage as an alternate mapper option. But the two codes are very different from each other, and connecting the two required us to deal with a number of challenges. This poster describes the codes, the challenges we worked through, current status, and some initial performance statistics.

ACM Reference Format:
Charles R. Ferenbaugh and Brendan K. Krueger. 2019. Linking a Next-Gen Remap Library into a Long-Lived Production Code. In *Proceedings of SC19: The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Eulerian Applications Project at Los Alamos National Laboratory (LANL) maintains xRage [4], an Eulerian AMR radiation-hydrodynamics code. xRage is descended from Sage, a code originally written around 1990. It contains about 470K lines of source code, not counting numerous third-party libraries from LANL and elsewhere. It is written mostly in Fortran 90, with some C and C++, and uses MPI-only parallelism. Efforts to modernize xRage to be more maintainable and adaptable to modern architectures started several years ago and are ongoing [2].

More recently, the Ristra next-generation code project has developed a remapping framework Portage [3]. Portage development started in 2015, and from the beginning it was designed to be highly modular and extensible, to support many different application contexts. Portage contains about 14K lines of source code. It is written in modern C++, and makes extensive use of classes and templates for customization. It also supports MPI+OpenMP parallelism, and GPU support is planned in the near future.

xRage contains a remapper capability that maps mesh fields from its native Adaptive Mesh Refinement (AMR) mesh to the Generalized Eulerian Mesh (GEM) format used by some third-party libraries (Figure 1). The current coding used for this capability was written in a short timeframe many years ago; this code is not well understood by the current xRage code team, and is difficult to maintain or extend. The xRage team has wanted to provide an alternate mapping capability, and has looked at Portage for this purpose. However the two codes are written in (mostly) different languages, have very different design philosophies, and are different in some of their underlying assumptions. This has led to several implementation challenges as we have worked to connect the two codes.

2 CHALLENGES

2.1 Legacy mapper cleanup

Like much of the original xRage code, the legacy mapper was written without much regard for modern software design principles. It was not encapsulated from the rest of the application; this would have made it difficult to know where and how to insert Portage as an alternative. It used private module data for all of its state variables; there was no distinction between GEM data and mapper data, which was problematic for Portage which needed the former

SC19, November 17-22, 2019, Denver, CO

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

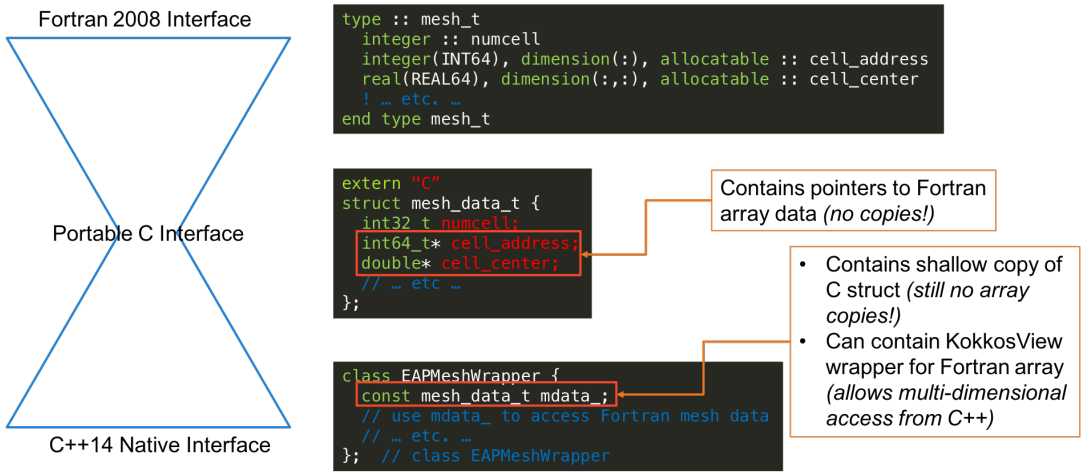

Figure 2: "Hourglass interface" pattern for interfacing between Fortran and C++.

but not the latter. And it was not unit-testable, but only tested in an integrated fashion with the physics packages that used it; this made initial xRage-Portage work difficult to test. Previous code modernization efforts had alleviated some of these issues, while the rest were handled by further refactoring done as part of the Portage integration effort.

2.2 Fortran/C++ interfaces

The Fortran code in the xRage application needed to call the C++ subroutines in the Portage framework, and to pass Fortran-declared data to be used in the C++ routines. To do this, we took advantage of the Fortran-to-C interoperability features introduced in Fortran 2003, and used the "Hourglass Interface" design pattern to pass Fortran-C++ communication through a narrow C interface that could be understood on both sides (Figure 2). We were able to pass pointers to Fortran data that could be accessed by C++ code without having to copy large arrays. Furthermore, we used KokkosView objects from the Kokkos library [1] to allow access to multidimensional Fortran arrays from the C++ side.

2.3 xRage extensions to Portage

In its early development, Portage made several assumptions that needed to be relaxed to support xRage. Portage supported cartesian coordinates only; xRage also must support cylindrical and spherical coordinates. Portage was written to support general unstructured meshes, and provided a general polygon/polyhedron intersector; xRage could restrict its support to axis-aligned boxes, so a more optimal box/box intersector was sufficient. Similarly, Portage included a kD-tree search/distribute method to handle general meshes; xRage could restrict its search method to GEM meshes, which allow for a more efficient search by decomposing the problem into k 1D searches. Fortunately the modular design of Portage made it easy to develop and use these extensions. They are currently in the xRage code base, but work is in progress to migrate them back into the Portage code base, to make them available to other Portage users.

3 RESULTS

An initial timing study has shown that the Portage mapper performs better than the legacy mapper, giving speedups of about 2x-22x depending on the problem type being run. A preliminary analysis also shows that Portage memory usage is about 7.6x lower than that of the legacy mapper.

While some details of the xRage-Portage remap are still being worked out, we are confident that this capability can be moved into production in the near future. Furthermore, this work will provide a model for more LANL next-generation packages to be integrated into production codes.

ACKNOWLEDGMENTS

Thanks to:

- Andrew Gaspar and Robert Pavel, for working through Fortran/C++ interface issues elsewhere in xRage
- Neil Carlson, for writing a Portage-Truchas link that provided some ideas for this work
- The xRage and Portage code teams

This work was supported by the US Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001).

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, 12 (2014), 3202 – 3216. https://doi.org/10.1016/j.jpdc.2014.07.003 Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[2] Charles Ferenbaugh, Sriram Swaminarayan, Chuck Aldrich, Matthew Calef, and Joann Campbell et al. 2016. Modernizing a Long-Lived Production Physics Code. In *Proceedings of SC16: The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC16).* https://http://sc16.supercomputing.org/sc-archive/tech_poster/poster_files/post196s2-file2.pdf

[3] Rao V. Garimella, Peter D. Crossman, Gary A. Dilts, Rajeev S. Erramilli, and Charles R. Ferenbaugh et al. 2017. PORTAGE - A Flexible Conservative Remapping Framework for Modern HPC Architectures. In *Proceedings of SC17: The International Conference for High Performance Computing, Networking, Storage, and*

Analysis (SC17). https://sc17.supercomputing.org/SC17%20Archive/tech_poster/tech_poster_pages/post169.html

[4] Michael Gittings, Robert Weaver, Michael Clover, Thomas Betlach, and Nelson Byrne et al. 2008. The RAGE radiation-hydrodynamic code. *Computational Science & Discovery* 1, 1 (2008), 015005.