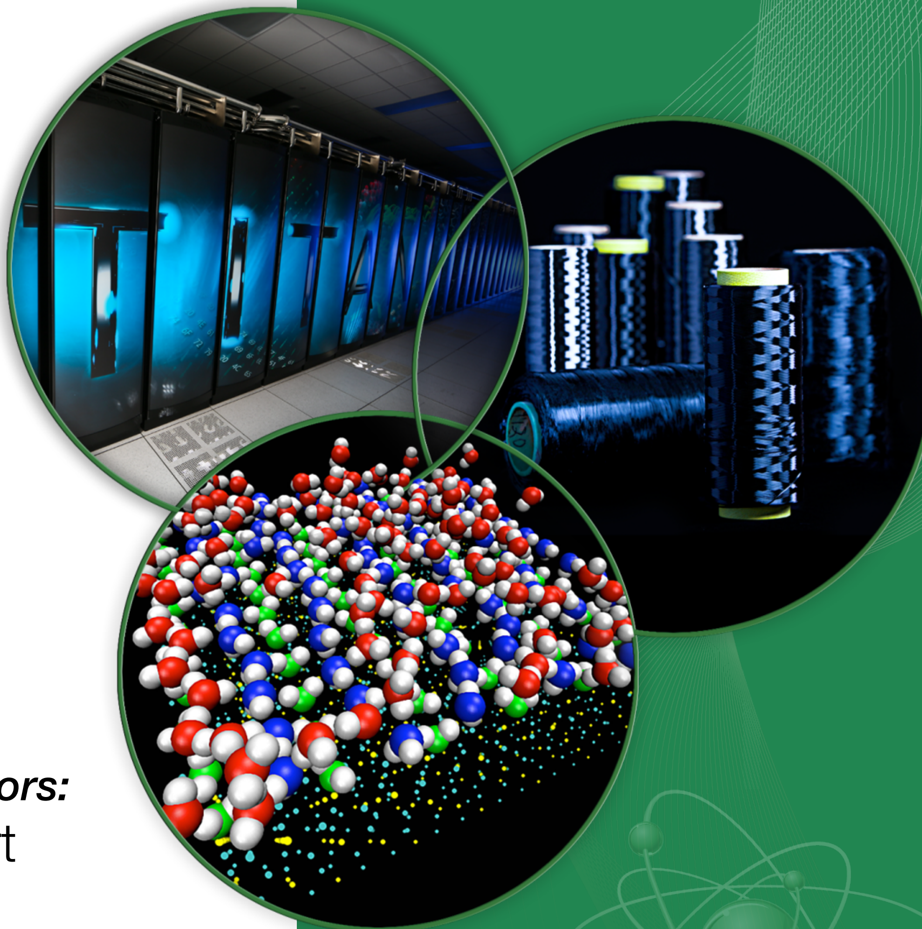
A first strike at an OpenACC C++ Monte Carlo code



Seth R Johnson, Ph.D.

R&D Staff, Monte Carlo Methods

Radiation Transport Group

Exnihilo team:

Greg Davidson

Tom Evans

Stephen Hamilton

Seth Johnson

Tara Pandya

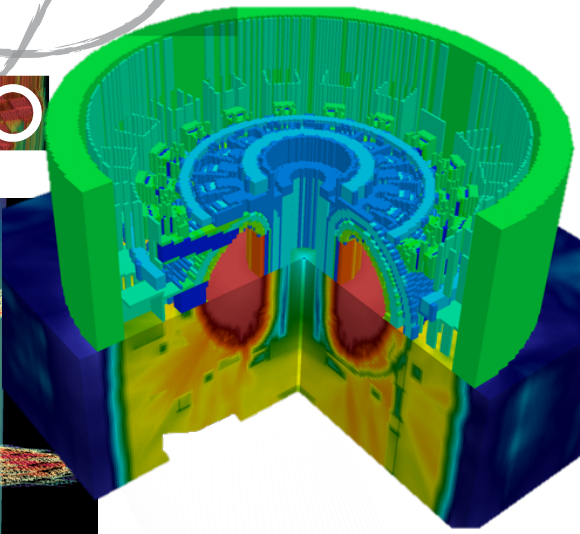
Hackathon Mentors:

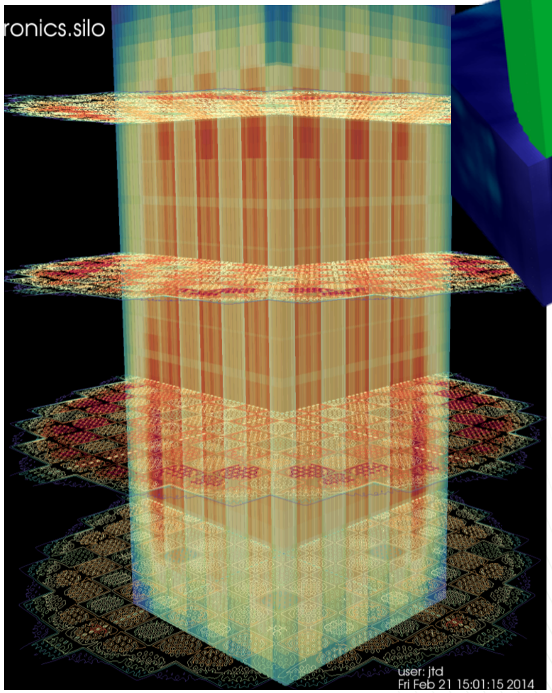
Wayne Joubert

Jeff Larkin

The codes

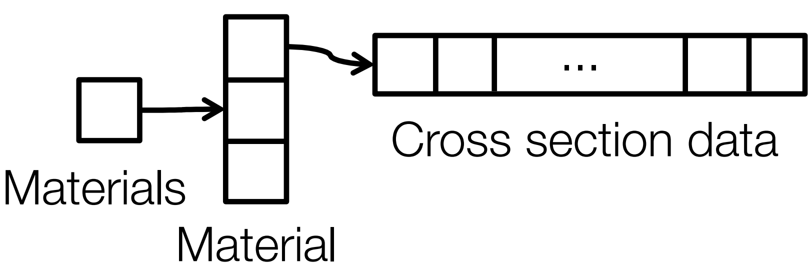
- Exnihilo: radiation transport framework
  - Multi-application (fusion, fission, detectors, homeland security)
  - Export controlled
- Profugus mini-app:
  - Written for algorithmic and HPC development
  - Limited capability
  - Reduced complexity

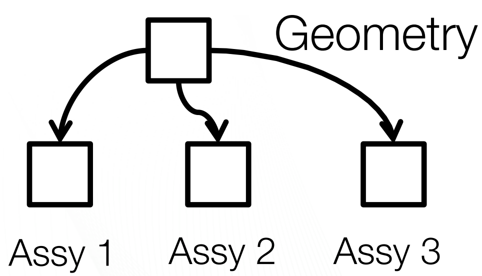




Introduction to the code environment

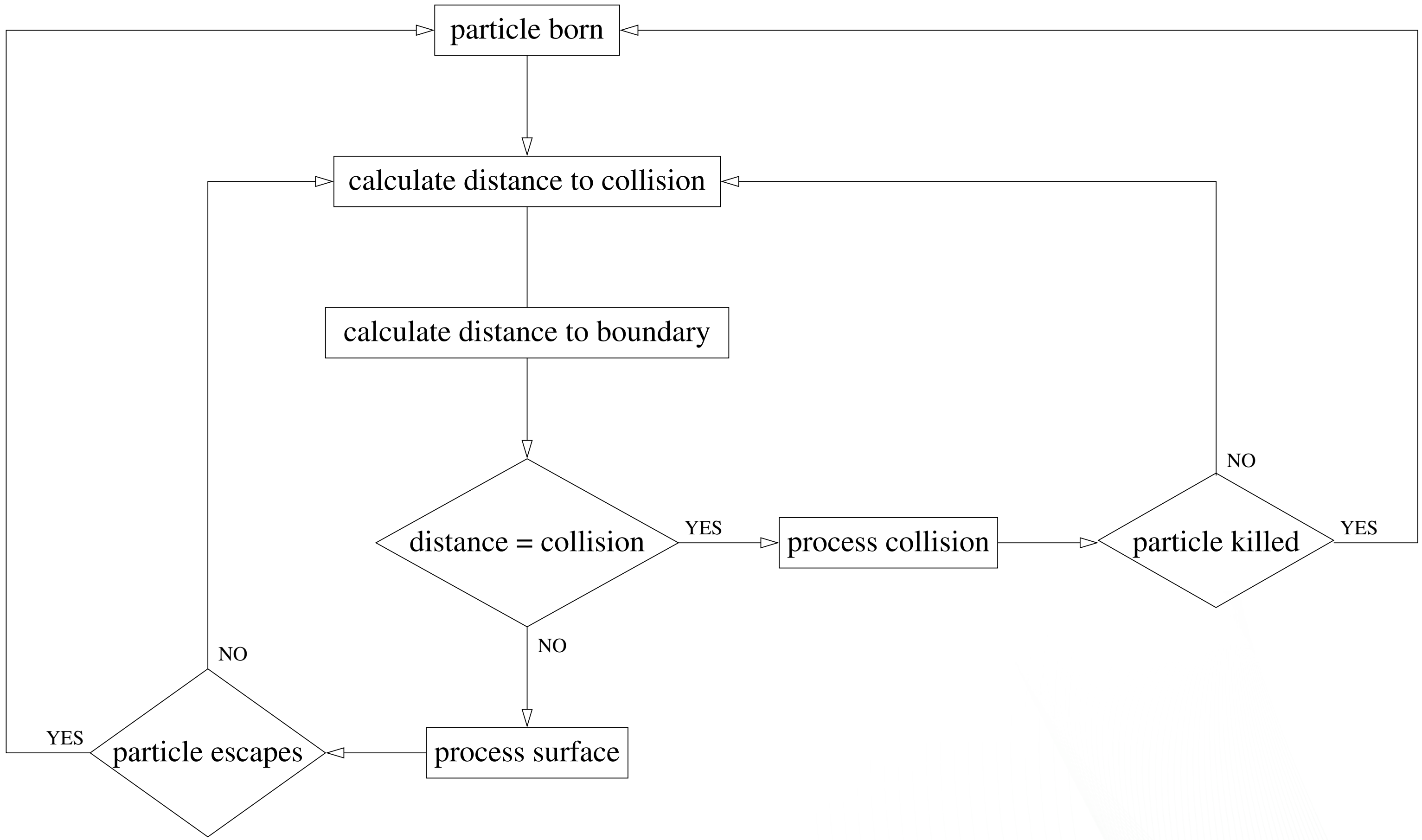
- C++11: unordered maps, `auto`, lambdas, etc.
- TPLs: Trilinos, HDF5
- Data structures are not POD, have irregular shape
  - Many distinct objects, dynamically sized vectors, shared pointers, etc. trade convenience for poorer data locality
  - Examples: particle, geometry cell, material attributes

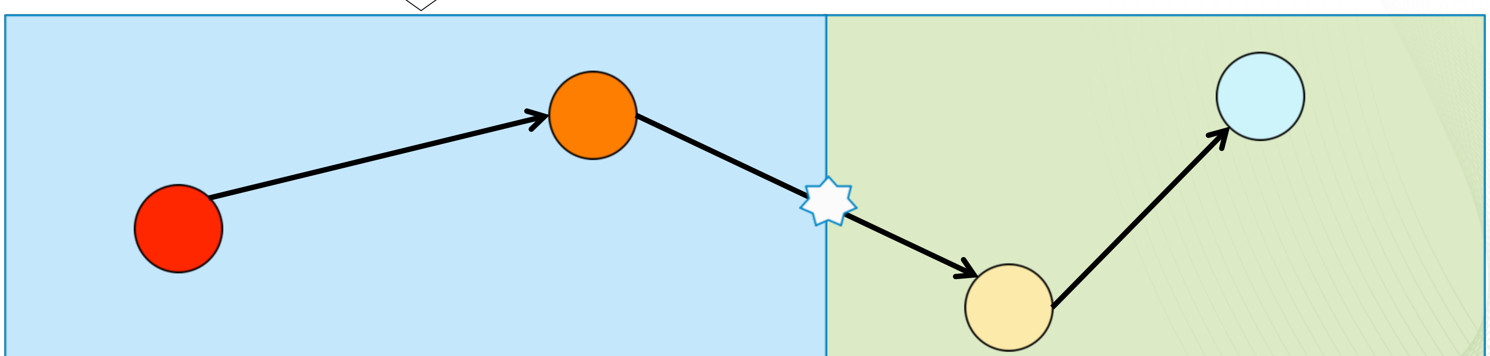




- Production environment: Chester (OLCF cluster)
  - PGI 14.7.0 (a few months old)
  - CUDA 5.5 (more than 2 years old)

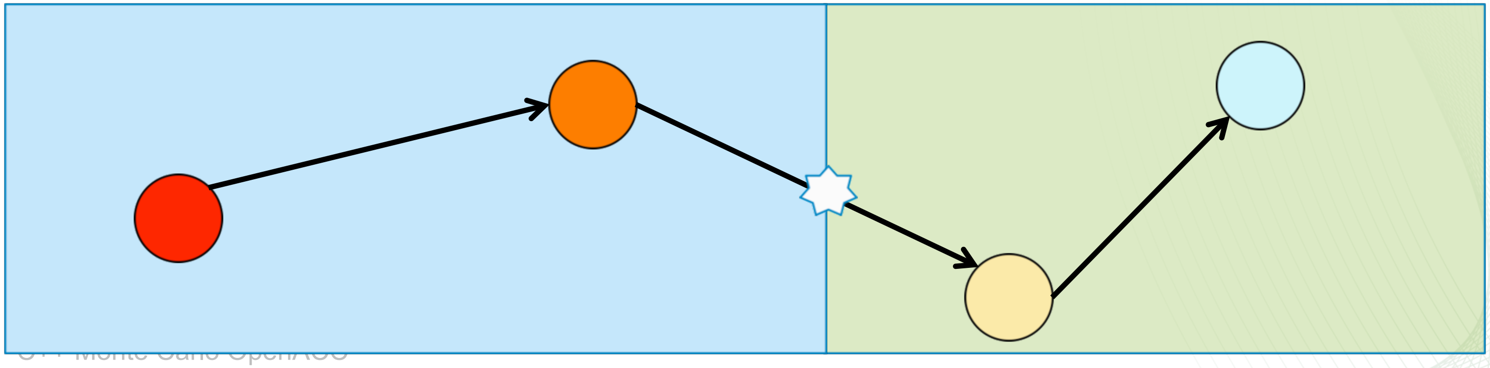
Introduction to Monte Carlo for neutronics





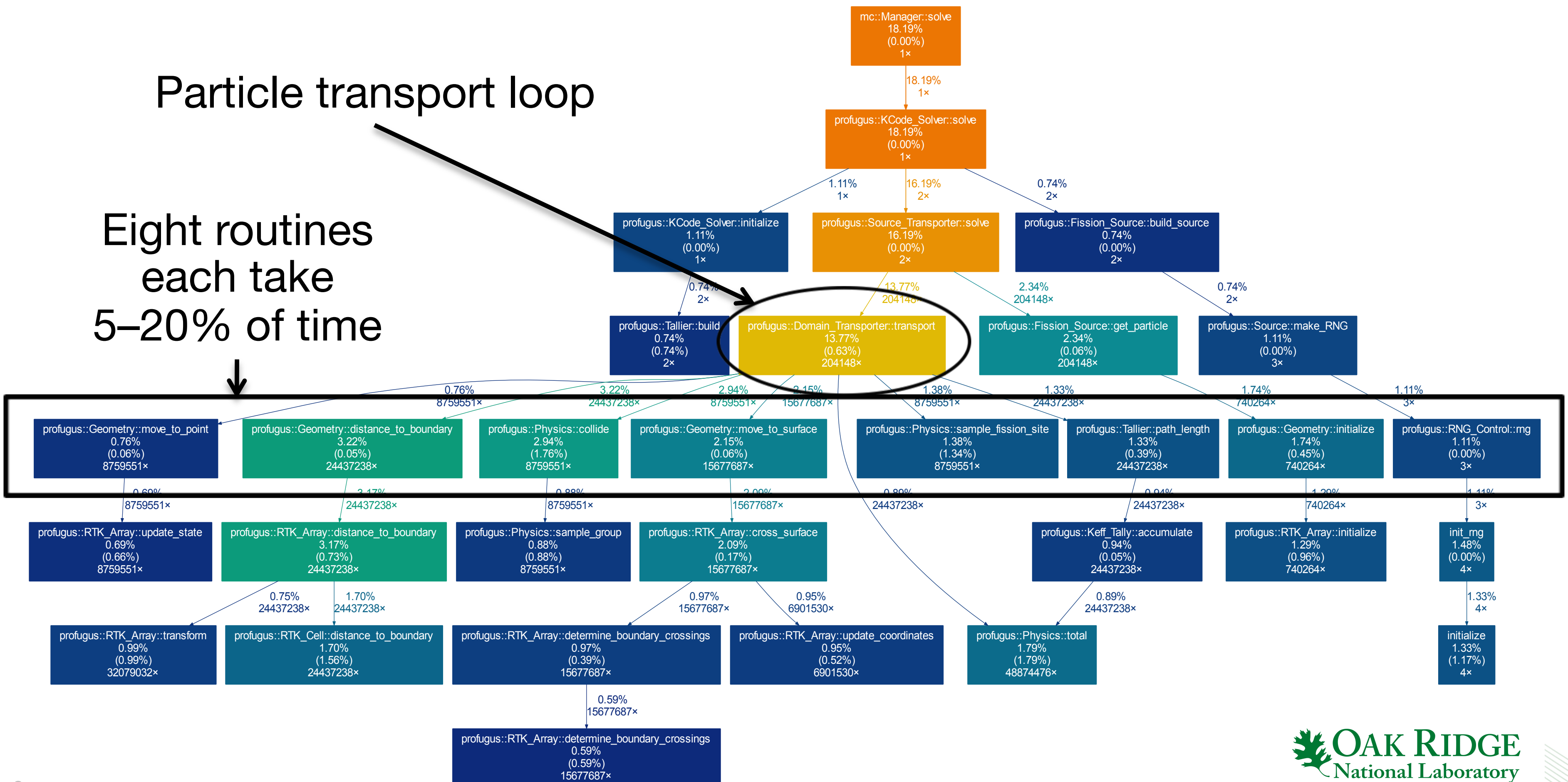
Algorithmic challenges

- Inherently stochastic process
  - Fast, long-period random number sampling required
  - Highly divergent code paths between loops
  - There is no fixed-length nested “`for` loop” to parallelize
- Complex data structures built to mirror physical processes
  - Indirection, dynamic allocation, irregular data shapes
  - There is no homogeneous multi-dimensional array of data



Initial timing profile

- Ran a semi-realistic reactor assembly problem
- No compute-intensive bottlenecks to offload



OAK RIDGE National Laboratory

The initial plan

- Rewrite classes for on-device execution
  - Geometry, Physics, Particle, Transporter
- Put CPU-intensive routines on the GPU
  - Particle geometry tracking
  - Cross section sampling and collisions
  - Tallying
- Run a simplified reactor assembly problem
- Get new timing profile using GPUs

OAK RIDGE National Laboratory

The immediate derailing of the initial plan

- Adding `-acc` flag broke our code
  - No OpenACC (or other) pragmas even being used
  - Unintelligible errors emitted from a standard library include inside Trilinos
  - Split OpenACC-dependent code into a subpackage that uses that flag, preventing its propagation elsewhere
- At least a day of team effort with Nvidia/PGI to get a C++ class with multiple vectors compiling

The final plan

- Attempt to write an adapter class to flatten CPU classes into data structures suitable for OpenACC
- Write a simple random number generator
- Write a simple brick mesh ray tracer that can be parallelized with OpenACC
- Write simple OpenACC-enabled multigroup physics with data access and collisions

What *actually* was accomplished

- 23 PGI compiler bug reports
  - PGI is the only compiler to support both OpenACC and C++11
  - We were probably the first group to use both in a production environment
- Primitive multigroup physics on the GPU
  - Driven through unit tests, reproduced CPU results
- Successfully ray-traced particles on brick mesh on the GPU
  - 20X faster if all particles do the same thing
  - 15X faster with divergence

OAK RIDGE National Laboratory

C++ suggestions for OpenACC

- Separate compilation units for ACC code
  - Inline keyword gives the compilers trouble; always write in .cc files
  - Include as few headers as possible (no Trilinos) to avoid compiler errors from non-ACC code and to reduce compiler time
- CPU data management with `std::vector`, then copy address to raw pointer for OpenACC
- Complexity hidden by ACC means more mysteries:
  - Do not rely on thread-private data

```
84, Accelerator restriction: scalar variable live-out from loop: seed
98, Loop carried scalar dependence for 'seed' at line 104
```

  - Issues with reduction operations on scalars
  - Do not use “const” class member data

Positive takeaways

- Learned basics of OpenACC and how it can be used in a C++ environment
- Better understanding of the heterogeneous architecture and how it relates to OpenACC directives (prior knowledge of CUDA is helpful)
- For very simplified and specific MC problems, we may be able to achieve speedup and the ability to run full problems on the GPU using Profugus (with a lot of rewriting)

Negative takeaways

- Existing MC algorithms are fundamentally incompatible with OpenACC-type usage
  - Monte Carlo does not have nested, fixed-length loops
  - Memory-managed objects cannot be accelerated
- C++, PGI, and OpenACC do not currently get along
  - Two weeks preparation to compile with PGI on Titan
  - C++11 incompatible with installed Cray compiler wrapper
  - Profiling tool issues with the code
  - Mystery compiler errors when turning on `-acc` on PGI
- No OpenACC libraries yet
  - We had write a simple pseudorandom number generator
  - No microkernels or algorithms for sorting, binary search

Concluding comments

- Hackathon was critical to kick-starting our investigation into Monte Carlo on the GPU
  - Resources: the compiler experts are there to help you
  - Time: you have a solid week to work in a focused environment with one task at hand
  - Perspective: you are not the only team struggling!
- OpenACC feasibility for C++
  - `#pragma` is not very pragmatic (inherently incompatible with C++ features): appropriate for Fortran
  - Compiler and environment are very difficult to get working
- Our next step: Kokkos as template-based abstraction layer

Acknowledgements

- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725
- Thanks to our mentors Jeff and Wayne (and to Matt) for their help!
- And thanks to Fernanda and OLCF for making the Hackathon happen!

Profugus:
http://ornl-cees.github.io/Profugus/

OAK RIDGE National Laboratory