

DOE HPCOR – ARDRA Project Overview

Adam J. Kunen kunen1@llnl.gov

September 15, 2015

LLNL-CONF-676935

Physics Domain

Deterministic (Sn) Particle Transport (Neutrons and Gamma Rays) on structured grids. We discretize and solve the Linear Boltzmann Transport Equation using the Discrete Ordinates method. Used for solving criticality, shielding and time-dependent problems.

Project Structure

4 Full-time developers, and 1 postdoc (all located at LLNL):

- Adam Kunen – Lead Computer Scientist
- Teresa Bailey – Nuclear Engineer
- Peter Brown – Mathematician
- Bujar Tagani – Computer Scientist
- Peter Maginot – Postdoc/Nuclear Engineer

~200k LOC, Mostly C++, some Fortran

~25 external library dependencies (ex. BLAS, LAPACK, Boost, HYPRE, Silo, Sundials, ...)

Mini App / Proxy App Activities

Since the ARDRA source code is Export Controlled, we developed an unrestricted proxy, KRIPKE. This has been valuable for external collaboration, both with student interns and with Sierra Center-of-Excellence.

Primary Methods

Numerical methods:

- Fully “upwind” parallel sweep algorithm – Sn transport specific
- Diffusion-Synthetic-Acceleration (DSA) uses HYPRE to solve large diffusion problems
- Power Iteration and Krylov solvers
- CFEM’s and DFEM’s for spatial discretizations
- Crank-Nicholson for time integration

Resources Typically Used

The range of interesting problems range from simple 1D problems running on 1 processor, all the way up to large complex 3D problems running on 1.5M cores of Sequoia (LLNL BG/Q machine).

Typically we run a lot of 16-core to 256-core problems on commodity x86-64 Linux clusters.

Exascale Preparation and Choices Made

At LLNL, we have had ongoing research into programming models to help provide performance portability. Most of this work has been done with the RAJA model, which allows for incremental refactoring of existing code bases with minimal code impact.

A major effort for ARDRA is to refactor its data structures to make it more amenable to RAJA-like abstractions. This includes making algorithms thread-safe, and transitioning from complex iterator-based data structures to array-based data.

We expect to be transitioning ARDRA from hand-coded C++ to the use of RAJA over the next year.

A major concern has been implementing efficient transport sweep algorithms on heavily threaded chips like the Intel Phi and the NVidia GPU's. Over that last year, we have had the Sierra COE aid in making transport sweeps efficient on a GPU in CUDA. This collaboration has been quite productive, and we now have a clear path forward on these platforms. Further work is needed to evaluate the ability of programming models to abstract the algorithms that were devised under this COE work.

In the past, we had a lot of success porting to the Sequoia machine. This presented us with a huge out-scaling problem. We were able to run a deterministic calculation on 37 trillion unknowns and 1.5M MPI ranks. This required us to resolve a number of solver and setup scalability issues. We are hoping that the work we are doing for Sierra, combined with the work we did with Sequoia, will enable us to move forward to an Exascale machine (assuming it looks like Sierra+Sequoia).

Major Issues with Software Stack

We continually have issues with software support.

Compiler optimizations have been a major impediment for programming model performance. Nvidia nvcc compiler support for C++ Lambda expressions is still very immature, and is needed for many of the abstractions we have examined (ex. RAJA and Kokkos).

Good debugging and profiling tools. We have current success with small scale (<256 MPI ranks) debugging with TotalView. We can do large scale debugging with LLNL's STAT tool.

NVidia has an excellent profiling tool "nvvp". We would love a tool like nvvp for CPU's. Also, debugging code written with programming model abstractions can be very difficult.

Tools for understanding memory and threads for CPUs are missing. For example, understanding how threads are being pinned to cores and how memory is pinned (or not) is important. Having better tools for understanding the memory access patterns for OpenMP codes is virtually non-existent. It can be very difficult to determine what a threaded code is actually doing at the hardware level. Most performance analysis that can be done is at a gross-level, and many assumptions or guesses must be made.