The XSBench Mini-App

John Tramm

April 3, 2013

Abstract

A kernel has been developed that mimics the most computationally expensive steps of a robust nuclear reactor core Monte Carlo particle transport algorithm – the calculation of macroscopic cross sections.

1 Algorithm

1.1 Reactor Model

When carrying out reactor core analysis, the geometry and material properties of a postulated nuclear reactor must be specified in order to define the variables and scope of the simulation model. For the purposes of this analysis, we use a well known community reactor benchmark known as the Hoogenboom-Martin model [1]. This model is a simplified analog to a more complete, "real-world" reactor problem, and provides a standardized basis for discussions on performance within the reactor simulation community. XSBench recreates the computational conditions present when fully featured MC neutron transport codes (such as OpenMC) simulate the Hoogenboom-Martin reactor model, preserving a similar data structure, a similar level of randomness of access, and a similar distribution of flops and memory loads.

1.2 Neutron Cross Sections

The purpose of an MC particle transport reactor simulation is to gain useful data about the distribution and generation rates of neutrons within a nuclear reactor. In order to achieve this goal, a large number of neutron lifetimes are simulated by tracking the path and interactions of a neutron through the reactor from its birth in a fission event to its escape or absorption, the latter possibly resulting in another fission event.

Each neutron in the simulation is described by three primary factors: its spatial location within a reactor's geometry, its speed, and its direction. At each stage of the transport calculation, a determination must be made as to what the particle will do next. For example, some possible outcomes include uninterrupted continuation of free flight, collision with another atom, or fission with a fissile material. The determination of which event occurs is based on a

random sampling of a statistical distribution that is determined by empirical material data stored in main memory. This data, called *neutron cross section data*, represents the probability that a neutron of a particular speed (energy) will undergo some particular interaction when it is inside a given type of material. To account for neutrons across a wide energy spectrum and materials of many different types, the structure that holds this cross section data must be very large. In the case of the simplified Hoogenboom-Martin benchmark roughly 5.6 GB[1] of data is required.

1.3 Data Structure

A nuclide is a type of atom described by a specific number of neutrons and protons. A given element can have many different nuclides, such as Uranium-235 or Uranium-238. A material in the reactor model is composed of a mixture of nuclides. For instance, the "reactor fuel" material might consist of several hundred different nuclides, while the "pressure vessel side wall" material might only contain a dozen or so. In total, there are 12 different materials and 355 different nuclides present in the modeled reactor.

For each nuclide, an array of nuclide grid points are stored as data in main memory. Each nuclide grid point has an energy level, as well as five various cross sections (corresponding to five different particle interaction types) for that energy level. The arrays are ordered from lowest to highest energy levels. Each nuclide has a different "grid spacing," i.e., the number, distribution, and granularity of energy levels varies between nuclides. One nuclide may have hundreds of thousands of grid points clustered around lower energy levels, while another nuclide may only have a few hundred grid points distributed across the full energy spectrum. This obviates straightforward approaches to uniformly organizing and accessing the data.

In order to increase efficiency of access, the algorithm utilizes another data structure, called the *unionized energy grid*, as described by Leppänen [2] and Romano [3]. The unionized grid facilitates fast lookups of cross section data from the nuclide grids. This structure is an array of grid points, consisting of an energy level and a set of pointers to the closest corresponding energy level on each of the different nuclide grids.

Table 1: XSBench Data Structure Summary

Nuclides Tracked	355
Total # of Energy Gridpoints	4,012,565
Cross Section Interaction Types	5
Total Size of Cross Section Data Structures	5.6 GB

[1] We estimate that for a robust depletion calculation in excess of 100 GB of cross section data would be required.

1.4 Access Patterns

In a full MC neutron transport application, the data structure is accessed each time a macroscopic cross section needs to be calculated. This happens anytime a particle changes energy (via a collision) or crosses a material boundary within the reactor. These macroscopic cross section calculations are extremely common in the MC transport algorithm, and the inputs to them are effectively random. For the sake of simplicity, XSBench was written ignoring the particle tracking aspect of the MC neutron transport algorithm and instead isolates the macroscopic cross section lookup kernel. This provides a large reduction in program complexity while retaining similarly random input conditions for the macroscopic cross section lookups via the use of a random number generator.

In XSBench, each macroscopic cross section lookup consists of two randomly sampled inputs: the neutron energy and the material. Given these two inputs, a binary (log n) search is done on the unionized energy grid for the given energy. Once the correct entry is found on the unionized energy grid, the material input is used to perform lookups from the nuclide grids present in the material. Use of the unionized energy grid means that binary searches do not need to be performed on each individual nuclide grid. For each nuclide present in the material, the two bounding nuclide grid points are found using the pointers from the unionized energy grid and interpolated to give the exact microscopic cross section at that point.

All calculated microscopic cross sections are then accumulated (weighted by their atomic density in the given material), which results in the macroscopic cross section for the material. The calculation of macroscopic cross sections is depicted in Algorithm 1.

Algorithm 1 Macroscopic Cross Section Lookup

1: $R(m_p, E_p)$ ▷ randomly sample inputs
2: *Locate* E_p *on Unionized Grid* ▷ binary search
3: **for** $n \in m_p$ **do** ▷ for each nuclide in input material
4: $\sigma_a \leftarrow n, E_p$ ▷ lookup bounding micro xs's
5: $\sigma_b \leftarrow n, E_p + 1$
6: $\sigma \leftarrow \sigma_a, \sigma_b$ ▷ interpolate
7: $\Sigma \leftarrow \Sigma + \rho_n \cdot \sigma$ ▷ accumulate macro xs
8: **end for**

In theory, one could "pre-compute" all macroscopic cross sections on the unionized energy grid for each material. This would allow the algorithm to run much faster, requiring far fewer memory loads and far fewer floating point operations per macroscopic cross section lookup. However, this would assume a static distribution of nuclides within a material. In practice, MC transport algorithms will need to track the burn-up of fuels, as well as heterogeneous temperature distributions within the reactor itself. This means that concentrations are dynamic, rather than static, therefore necessitating the use of the more versatile data model deployed in XSBench.

We have verified that XSBench faithfully mimics the data access patterns of the full MC application under a broad range of conditions. The runtime of full-scale MC transport applications, such as OpenMC, is 85% composed of macroscopic cross section look ups. Within this process, XSBench is virtually indistinguishable, as the same type and size of data structure is used, with a similarly random access pattern and a similar number of floating point operations occurring between memory loads. Thus, performance analyses done with XSBench will provide results applicable to the full MC neutron transport algorithm, while being easier to interpret.

2 Performance

XSBench has been tested on a single node of the IBM BlueGene/Q supercomputer *Mira* at Argonne National Laboratory, as well as a single node intel Xeon E5620 system containing 8 cores (dual quad-core processors), each with 2 hardware threads.

Each system (BlueGene/Q and Xeon) was tested with between 1 and its maximum number of supported threads (64 for BG/Q, 16 for the intel Xeon machine). A set number of macroscopic cross section calculations were performed for each run, and the speed of completion, in terms of cross section lookups per second, was recorded. The results can be seen in Figures 1 and 2.

Another interesting result from our experiment was further confirmation of the cause of bottlenecking in the MC particle transport algorithm on current generation systems. Since the addition of hardware threads caused significant increases in performance, it is clear that the algorithm is not limited by memory bandwidth and is instead likely limited by memory latency. If the algorithm were limited by bandwidth, addition of extra hardware threads would show no performance increase, as the additional memory load requests streaming off the hardware threads would have to contend with an already saturated pipeline. However, the observed performance is consistent with a scenario in which memory latency limits the speed of the calculation. The addition of hardware threads allows for an increase in the total number of memory loads being generated, each with a static latency associated with it. As a higher volume of these can exist at the same time (due to the hardware threads), the calculation completes at an increased rate.

3 Future Work

In future work, we plan to develop useful metrics based on performance counters in XSBench in order to determine precisely where and how hardware performance bottlenecks in the kernel are caused. Similarly, we will investigate exactly why there are diminishing returns when adding multiple hardware threads to each CPU. Furthermore, there are additional capabilities that do not yet exist in full-scale MC neutron transport algorithms, such as on-the-fly doppler broad-

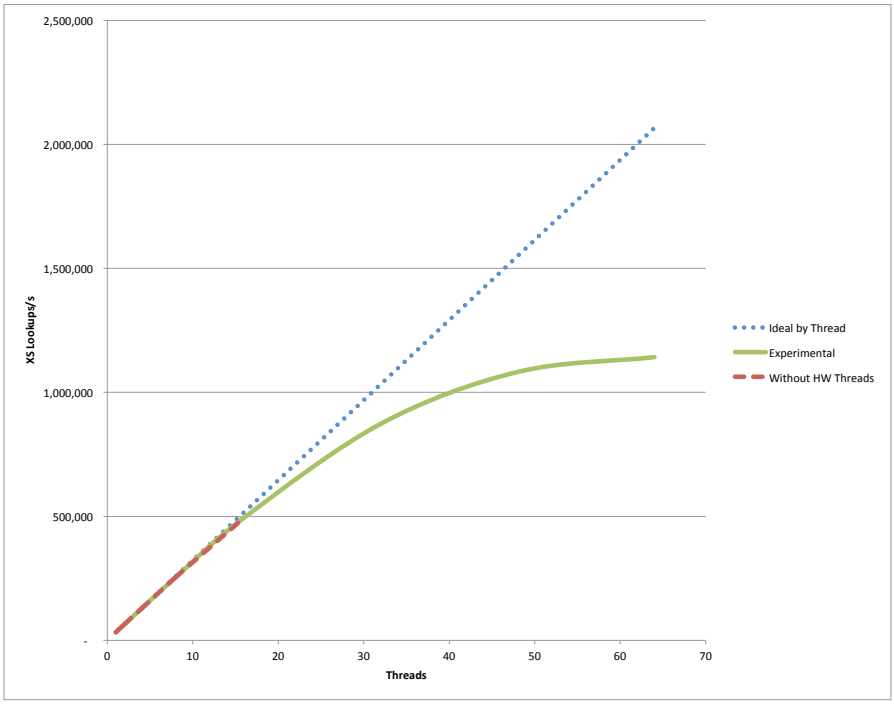

Figure 1: Scaling on BlueGene/Q

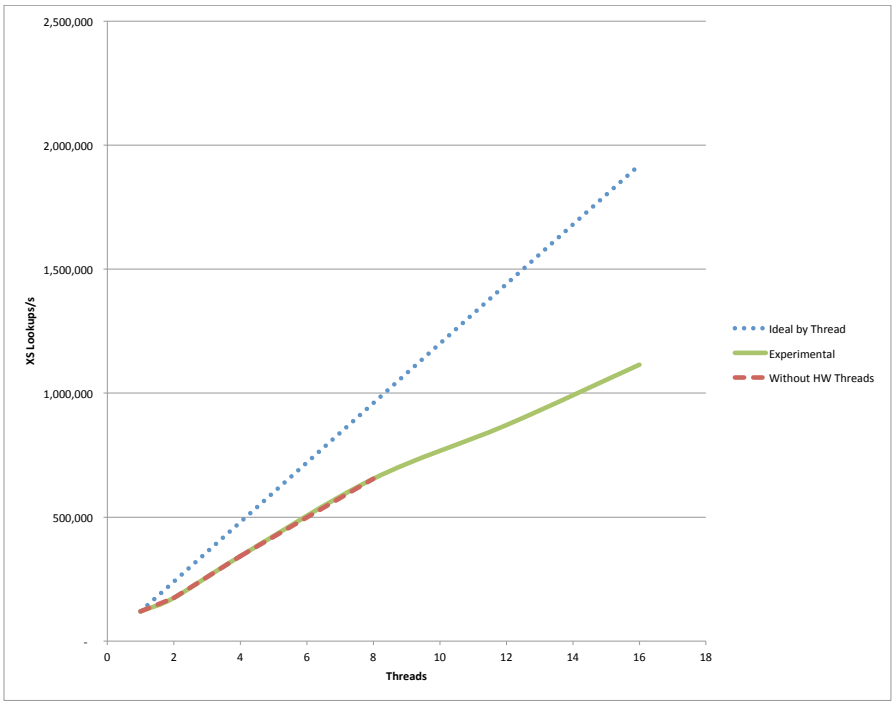

Figure 2: Scaling on Xeon

ening to account for the material temperature dependence of cross sections, that we plan on adding to XSBench for experimentation with various hardware architectures and features.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE- AC02-06CH11357. The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (Argonne) under Contract DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

References

[1] J.E. Hoogenboom, W.R. Martin, B. Petrovic, "The Monte Carlo performance benchmark test aims, specifications and first results," International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Rio de Janeiro, Brazil (2011).

[2] Jaakko Leppänen, "Two practical methods for unionized energy grid construction in continuous-energy Monte Carlo neutron transport calculation," Annals of Nuclear Energy, Volume 36, Issue 7, July 2009, Pages 878-885.

[3] Paul K. Romano, Benoit Forget, "The OpenMC Monte Carlo particle transport code," Annals of Nuclear Energy, Volume 51, January 2013, Pages 274-281.

[4] Paul K. Romano, Benoit Forget, Forrest Brown, "Towards Scalable Parallelism in Monte Carlo Particle Transport Codes Using Remote Memory Access," Progress in Nuclear Science and Technology, Volume 2, October 2011, Pages 670-675.

[5] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra, R. Biswas, The impact of hyper-threading on processor resource utilization in production applications, 18th International Conference on High Performance Computing (HiPC), Bangalore, India, Dec. 2011.

[6] Andrew Siegel, Kord Smith, Paul Romano, Ben Forget, Kyle Felker, "Multicore performance studies of a Monte Carlo neutron transport code," International Journal of High Performance Computing Applications. (Under Review)

[7] John Tramm, Andrew Siegel, "Memory Bottlenecks and Memory Contention in Multi-Core Monte Carlo Transport Codes," Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo, 2013. (Submitted, Pending Acceptance & Review. Not yet Published.)