Annals of Nuclear Energy 82 (2015) 195–202

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes

John R. Tramm *, Andrew R. Siegel

Argonne National Laboratory, Center for Exascale Simulation of Advanced Reactors, 9700 S Cass Ave, Argonne, IL 60439, United States

ARTICLE INFO

Article history:
Received 30 April 2014
Accepted 19 August 2014
Available online 4 September 2014

Keywords:
Monte Carlo
Neutron transport
Reactor simulation
Cross section
Benchmarks
High performance computing

ABSTRACT

We have extracted a kernel that executes only the most computationally expensive steps of the Monte Carlo particle transport algorithm – the calculation of macroscopic cross sections – in an effort to expose bottlenecks within multi-core, shared memory architectures.

1. Introduction

Current and next generation processor designs require exploiting on-chip, fine-grained parallelism to achieve a significant fraction of theoretical peak CPU speed. The success or failure of these designs will have a tremendous impact on the performance and scaling of a number of key reactor physics algorithms run on next-generation computer architectures. One key example is the Monte Carlo (MC) method for neutron transport. MC methods are characterized by complex memory access patterns that heavily tax shared resources of multi-core memory hierarchies. In this analysis we study in depth the on-node scaling properties and memory contention issues of MC particle transport specifically for reactor physics calculations.

There has been significant research into the performance and scaling of MC particle transport algorithms on distributed memory, High Performance Computing (HPC) systems (Romano and Forget, 2013; Romano et al., 2011). One such effort, the *OpenMC* transport code (Romano and Forget, 2013), has investigated scaling on many-node distributed memory architectures, such as Blue Gene /P. However, there is little reported on the performance of such applications on multi-core, shared memory architectures. At least one recent study does provide a comprehensive view of on-node scaling behavior at the algorithmic level but does not go into great depth on the underlying architectural causes of scaling degradation (Siegel et al., 2014). This is notable as there are significant memory contention issues in multi-core scaling that are not present on distributed memory architectures. Typical (Siegel et al., 2014) multi-core scaling for the MC particle transport algorithm is shown in Fig. 1.

To investigate scaling and performance issues of robust, quasi-static nuclide depletion calculations (i.e., where hundreds of nuclides are present in the fuel region and performance is dominated by macroscopic cross section calculations), such as are performed by *OpenMC*, we abstract a key computational kernel that is responsible for the majority of the algorithm's runtime and implement it in the form of the "proxy application" *XSBench*. The end result is that the essential computational conditions and tasks of fully featured MC transport codes are retained in the kernel, without the additional complexity of the full application. This provides a much simpler and more transparent platform for isolating where both hardware and software bottlenecks inhibit scaling of the algorithm. We then use and modify our extracted kernel to identify low-level hardware and software bottlenecks on an Intel Xeon system, so that we can make an intelligent prediction as to how the MC transport algorithm will scale on next generation, many-core systems.

1.1. The reactor simulation problem

Computer-based simulation of nuclear reactors is a well established field, with origins dating back to the early years of digital

* Corresponding author. Tel.: +1 847 421 1534.
E-mail addresses: jtramm@mcs.anl.gov (J.R. Tramm), siegel a@mcs.anl.gov (A.R. Siegel).

http://dx.doi.org/10.1016/j.anucene.2014.08.038
0306-4549/

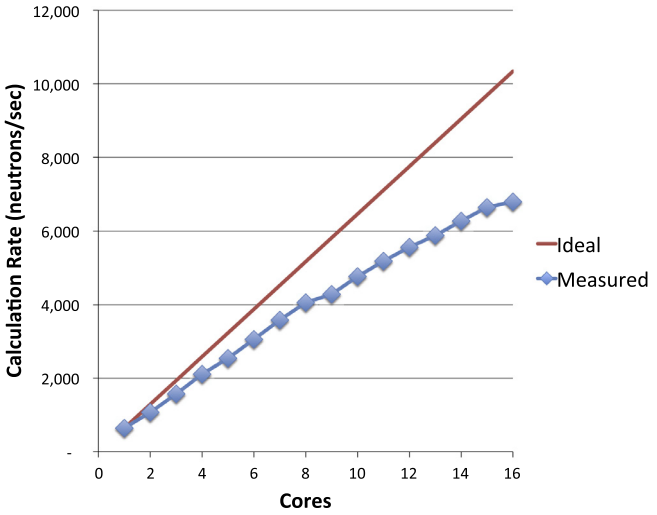

Fig. 1. *OpenMC* performance scaling on a 16-core xeon node.

computing. Traditional reactor simulation techniques aim to solve the diffusion equation for a given material geometry and starting (source term) neutron distribution within the reactor. This is done in a deterministic fashion using well developed numerical methods. Deterministic codes are capable of running quickly and providing precise solutions, however, there are other approaches to the problem that offer potential advantages.

An alternative method, Monte Carlo (MC) simulation, simulates the path of a particle neutron as it travels through the reactor core. As many particle histories are simulated, a picture of the full distribution of neutrons within the reactor core is developed. Such codes are inherently simple, easy to understand, and potentially easy to rethink when moving to new, novel architectures. Furthermore, the methodologies utilized by MC simulation require very few assumptions, resulting in highly accurate results assuming adequate statistical convergence. The downside to this method, however, is that a huge number of neutron histories must be run in order to achieve an acceptably low variance in the results. For many problems this means an impractically long time-to-solution, though such limitations may be overcome given the increased computational power of next-generation, exascale supercomputers.

1.2. OpenMC

OpenMC is a Monte Carlo particle transport simulation code focused on neutron criticality calculations (Romano and Forget, 2013). It is capable of simulating 3D models based on constructive solid geometry with second-order surfaces. The particle interaction data is based on ACE format cross sections, also used in the MCNP and Serpent Monte Carlo codes. *OpenMC* has been used to investigate scaling concerns on distributed memory architectures, such as the IBM Blue Gene/P and Blue Gene/Q.

OpenMC was originally developed by members of the Computational Reactor Physics Group at the Massachusetts Institute of Technology starting in 2011. Various universities, laboratories, and other organizations (including CESAR) now contribute to the development of *OpenMC*.

1.3. XSBench

The *XSBench* proxy application models the most computationally intensive part of a typical MC transport algorithm – the calculation of macroscopic neutron cross sections, a kernel which accounts for around 85% of the total runtime of *OpenMC* (Siegel et al., 2014). *XSBench* retains the essential performance-related computational conditions and tasks of fully featured reactor core MC neutron transport codes, yet at a fraction of the programming complexity of the full application. Particle tracking and other features of the full MC transport algorithm were left out of *XSBench* as they take up only a small portion of runtime. This provides a much simpler and far more transparent platform for testing the algorithm on different architectures, making alterations to the code, and collecting hardware runtime performance data.

XSBench is in active development by members of the Center for Exascale Simulation of Advanced Reactors (CESAR) at Argonne National Laboratory. The application is written in C, with multi-core parallelism support provided by OpenMP. *XSBench* is an open source software project. All source code is publicly available online.

2. Algorithm

2.1. 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 *XSBench*, we use a well known community reactor benchmark known as the Hoogenboom-Martin model (Hoogenboom et al., 2010). 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.

2.2. Neutron cross sections

The purpose of an MC particle transport reactor simulation is to calculate 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 subsequent fission events.

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. Possible outcomes include uninterrupted continuation of free flight, collision, or absorption (possibly resulting in fission). The determination of which event occurs is based on a random sampling of a statistical distribution that is described 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 data 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.

2.3. Data structure

A material in the reactor model is composed of a mixture of nuclides. For instance, the "reactor fuel" material might consist

[1] We estimate that for a robust depletion calculation, in excess of 100 GB of cross section data would be required (Romano et al., 2013).