XSBENCH – THE DEVELOPMENT AND VERIFICATION OF A PERFORMANCE ABSTRACTION FOR MONTE CARLO REACTOR ANALYSIS

John R. Tramm and Andrew R. Siegel
Center for Exascale Simulation of Advanced Reactors
Argonne National Laboratory, Argonne, IL, USA
jtramm@mcs.anl.gov
siegela@mcs.anl.gov

Tanzima Islam and Martin Schulz
Lawrence Livermore National Laboratory, Livermore, CA, USA
islam3@llnl.gov
shulzm@llnl.gov

ABSTRACT

We isolate the most computationally expensive steps of a robust nuclear reactor core Monte Carlo particle transport simulation. The *hot kernel* is then abstracted into a simplified *proxy application*, designed to mimic the key performance characteristics of the full application. A series of performance verification tests and analyses are carried out to investigate the low-level performance parameters of both the simplified kernel and the full application. The kernel's performance profile is found to closely match that of the application, making it a convenient test bed for performance analyses on cutting edge platforms and experimental next-generation high performance computing architectures.

Key Words: **XSBench, OpenMC, Monte Carlo, neutron transport, multi-core, reactor simulation**

1. INTRODUCTION

Monte Carlo (MC) transport algorithms are considered the "gold standard" of accuracy for a broad range of applications – e.g., nuclear reactor physics, shielding, detection, medical dosimetry, and weapons design to name just a few examples. In the design and analysis of nuclear reactor cores, the key application driver of the present analysis, MC methods for neutron transport offer significant potential advantages compared to deterministic methods given their simplicity, avoidance of ad hoc approximations in energy treatment, and lack of need for complex computational meshing of reactor geometries.

On the other hand it is well known that robust analysis of a full reactor core is still beyond the reach of MC methods. Tremendous advances have been made in recent years, but the computing requirements for full quasi-static depletion analysis of commercial reactor cores is a performance-bound problem, even on existing leadership class computers. It is also clear that many of the issues related to scalability on distributed memory machines have been adequately addressed in recent studies[1][2], and that the path to future speedups involves taking better advantage of a broad range of multi-core systems.

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 the neutron transport application *OpenMC*[1], 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 run a series of analyses that investigate the low-level performance parameters of both our proxy application, *XSBench*, and the full scale application, *OpenMC*, that it seeks to mimic. The performance profiles are compared to determine the accuracy of *XSBench* in recreating the computational conditions of *OpenMC* and to determine its suitability as a "stand-in" for the full application. This study is done so that future analyses can be done in which we use and modify our extracted kernel to identify low-level hardware and software bottlenecks, 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 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 given 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 [1]. 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 now contribute to its development. The application is written in FORTRAN, with parallelism supported by a hybrid OpenMP/MPI model. *OpenMC* is an open source software project available online[3].

1.3. XSBench

The *XSBench* proxy application models the most computationally intensive part of a typical MC reactor core transport algorithm – the calculation of macroscopic neutron cross sections, a kernel which accounts for around 85% of the total runtime of *OpenMC*[4]. *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 not included in *XSBench* as they take up only a small portion of runtime in robust reactor computations. 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 was developed 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[5].

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[6]. 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 algorithm requires use of a very large data structure that holds cross section data points for many discrete energy levels. 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 Hoogenboom-Martin 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. The data usage requirements to store this model are significant, totaling 5.6 GB, as summarized in Table I.

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 cross section values (corresponding to five different particle interaction types) for that energy level. The arrays are ordered from lowest to highest energy levels. 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.

[1] We estimate that for a robust depletion calculation, in excess of 100 GB of cross section data would be required.[7]

In order to increase efficiency, the algorithm utilizes another data structure, called the *unionized energy grid*, as described by Leppänen[8] and Romano[1]. 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.

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

Table I. *XSBench* Data Structure Summary

2.4. 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 occur with very high frequency 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 E_p, and the material m_p. 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 are not required 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. Algorithm 1 is an abbreviated summary of this calculation.

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 nuclide-depletion calculations are quasi-static; they will need to track the burn-up of fuels and account for 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 *OpenMC* and *XSBench*. Even if static concentrations were assumed, pre-computation of the full

Algorithm 1 Classical Continuous Energy Macroscopic Cross Section Lookup

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

spectrum of macroscopic cross sections would need to be done for all geometric regions (of which there are many millions) in the reactor model, leading to even higher memory requirements.

3. EXPERIMENTS

To investigate the performance and resource utilization profiles of both applications, and to determine the similarity in performance parameters between *XSBech* and *OpenMC*, we performed a series of experiments. Each experiment involves monitoring specific aspects of hardware usage using performance counters. The following section presents descriptions, results, and preliminary conclusions for each experiment. For the purposes of simplicity, we concentrate our analysis on a single node, multi-core, shared memory system. The system used was a PC node consisting of two Intel Xeon octo-core CPUs for a total of 16 physical CPUs[2]. This allows us to get highly in-depth results as we are able to run experiments dealing with architecture-specific features and hardware counters.

In order to determine the specific hardware profiles of *XSBench* and *OpenMC*, both applications are instrumented to collect hardware performance counter data with the Performance Application Programming Interface (PAPI)[9].

3.1. Multi-Core Scaling Efficiency

The first step in our performance analysis is to investigate the basic performance scaling behaviour of *OpenMC* and *XSBench*. We ran both applications with only a single thread to determine a baseline performance against which efficiency can be measured. Then, further runs were done to test each number of threads between 1 and 32. Efficiency is defined in Equation 1, where n is the number of cores, R_n is the experimental calculation rate for n cores, and R_1 is the experimental calculation rate for one core.

$$\text{Efficiency}_n = \frac{R_n}{R_1 \times n} \tag{1}$$

[2]The 16-core Xeon node used in our testing features hardware threading, supporting up to 32 threads per node.

The tests reveal that even for this idealized representation of the key MC transport algorithm, perfect scaling was not achievable. Figure 1 shows that efficiency degraded gradually as more cores were used on the nodes. Efficiency at 16 threads degraded to 66% in *OpenMC*, and 69% in *XSBench*. As this is a difference of only 4.5%, *XSBench* can be considered a very accurate tool for representing multi-core scaling efficiency of the Monte Carlo neutron transport algorithm.

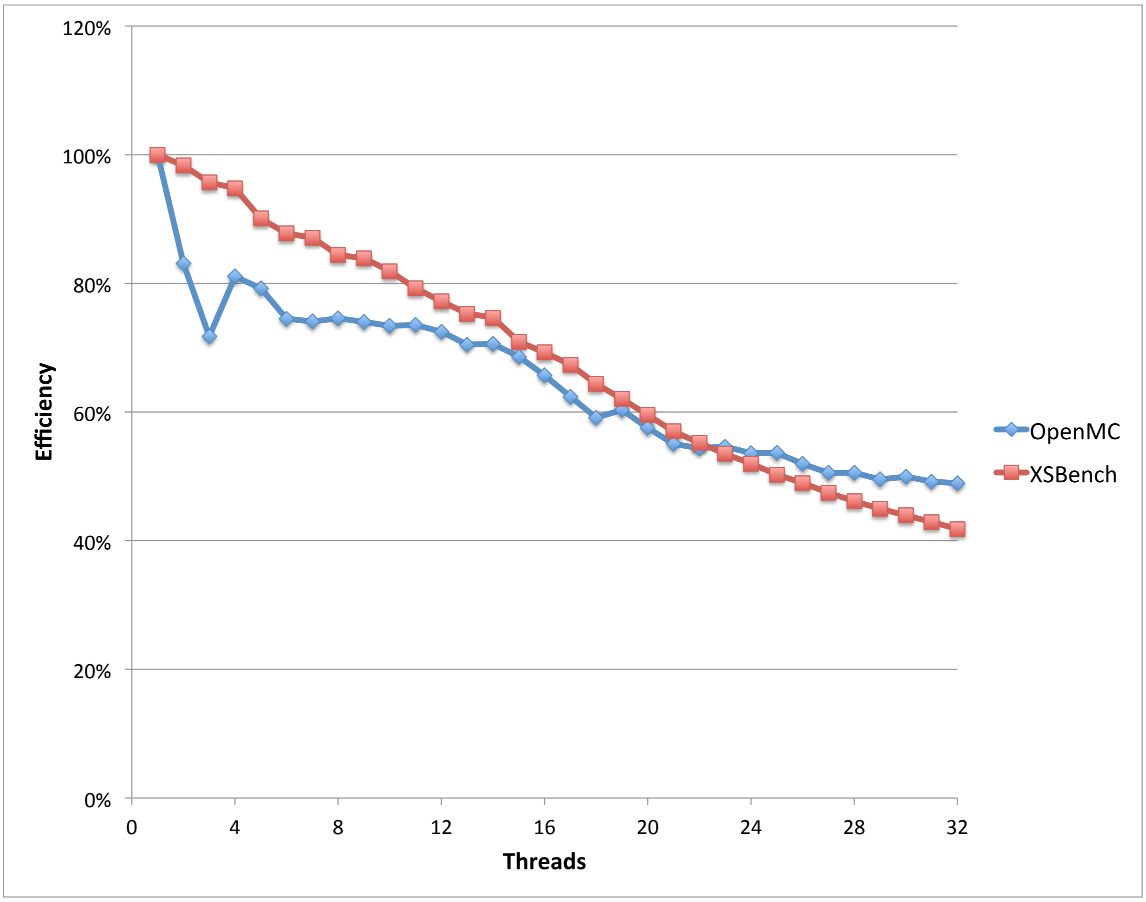

Figure 1. Efficiency Scaling

3.2. Floating Point Calculation Rate

Consumption of available system floating point resources used by *XSBench* and *OpenMC* is calculated using Equation 2, where `PAPI_FP_INS` and `PAPI_TOT_CYC` are hardware counter values that represent the total number of floating point instructions retired and total number of hardware cycles used by the programs respectively. The clockspeed of our Xeon test system is 2.8 GHz.

$$\text{FLOPs} = \frac{\text{PAPI\_FP\_INS}}{\text{PAPI\_TOT\_CYC}} \times \text{Clock (Hz)} \tag{2}$$

Using Equation 2, we were able to determine the FLOP performance of our applications, as shown in Figure 2. We found that *XSBench* achieved at most 4.2 GFLOPs, while *OpenMC* achieved at most 4.4 GFLOPs. As this is a difference of only 4.4%, *XSBench* can be considered an accurate tool for measuring the floating point calculation rate of the Monte Carlo neutron transport algorithm.

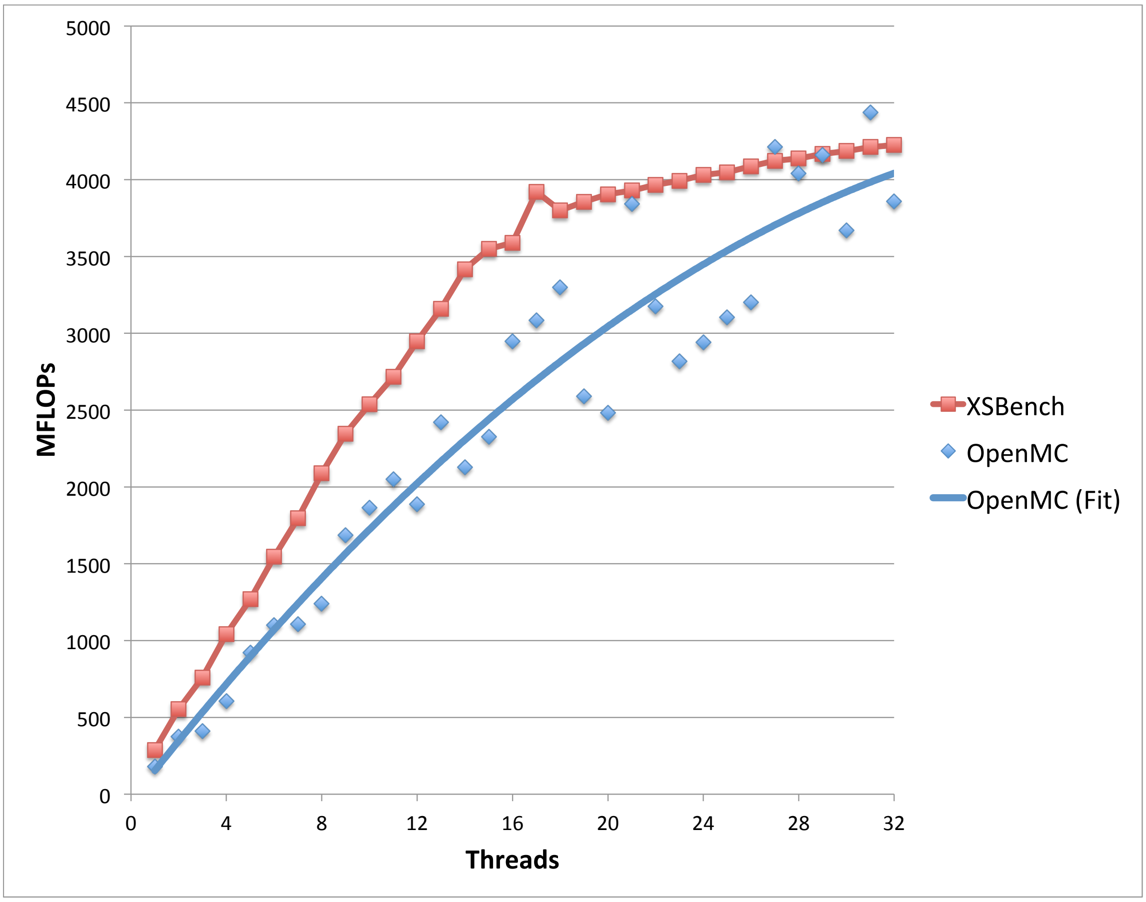

Figure 2. FLOP Usage

3.3. General Hardware Performance Counter Profiles

To further investigate the performance models of our two applications beyond the obvious parameters, we conducted the following steps to measure a broad spectrum of performance correlations:

Step 1: For both *XSBench* and *OpenMC*, we collected 47 hardware performance counters for a code region that was identified as the region of interest. In the case of *OpenMC*, this region was the full particle transport loop. In the case of *XSBench*, this was the cross section lookup loop. We collected these counters for runs that used between 1 and 15 threads with 1 thread per core on an Intel Xeon machine.

Step 2: Equation 3 shows how we computed the correlation coefficients between the values of each counter for threads 1 through 15 with their corresponding efficiency loss. Each correlation value indicates how the value of a performance counter correlates to the efficiency loss for all thread cases. A positive strong correlation indicates that this performance counter captures the event that may have some role in the loss of efficiency for this application. Figure 3 shows these computed correlations for both *XSBench* and *OpenMC* on the Intel Xeon.

$$\begin{aligned} \%\ \text{efficiency\_loss}_j &= \left(1 - \frac{\text{runtime}_1}{\text{runtime}_j * j}\right) \times 100 \\ \text{correlation\_coeff}[i] &= \text{pearson\_correlation}\left(\text{counter}_j[i], \text{efficiency\_loss}_j\right) \\ &\forall i \in \{1, 2, \ldots, \text{Number of performance counters}\} \\ &\forall j \in \{1, 2, \ldots, \text{Number of threads}\} \end{aligned} \tag{3}$$

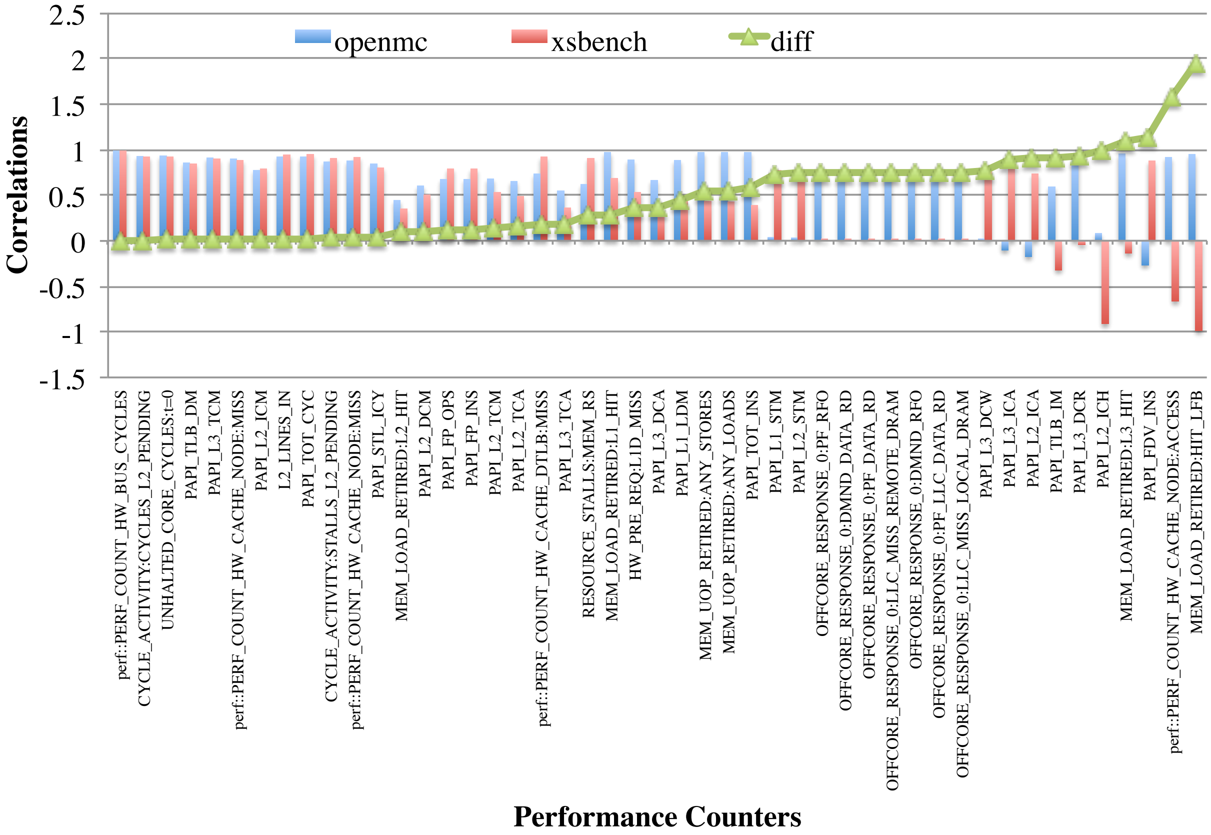

Figure 3. Correlation of all performance variables

Step 3: Figure 4 shows a handpicked short list of performance counters that have ratios greater than 0.85, which demonstrates how effective the proxy application can emulate the resource usage characteristics of its full application counterpart. From Figure 4, we can observe that all 6 performance counters that most directly affect the loss of efficiency during scaling for both *OpenMC* and *XSBench* are counters for events that contend for physical resources in the memory system such as

L2 cache, L3 cache, and memory bus – resources that are shared across cores. This result indicates that *XSBench* is a good proxy application for emulating the memory system behavior of *OpenMC*.

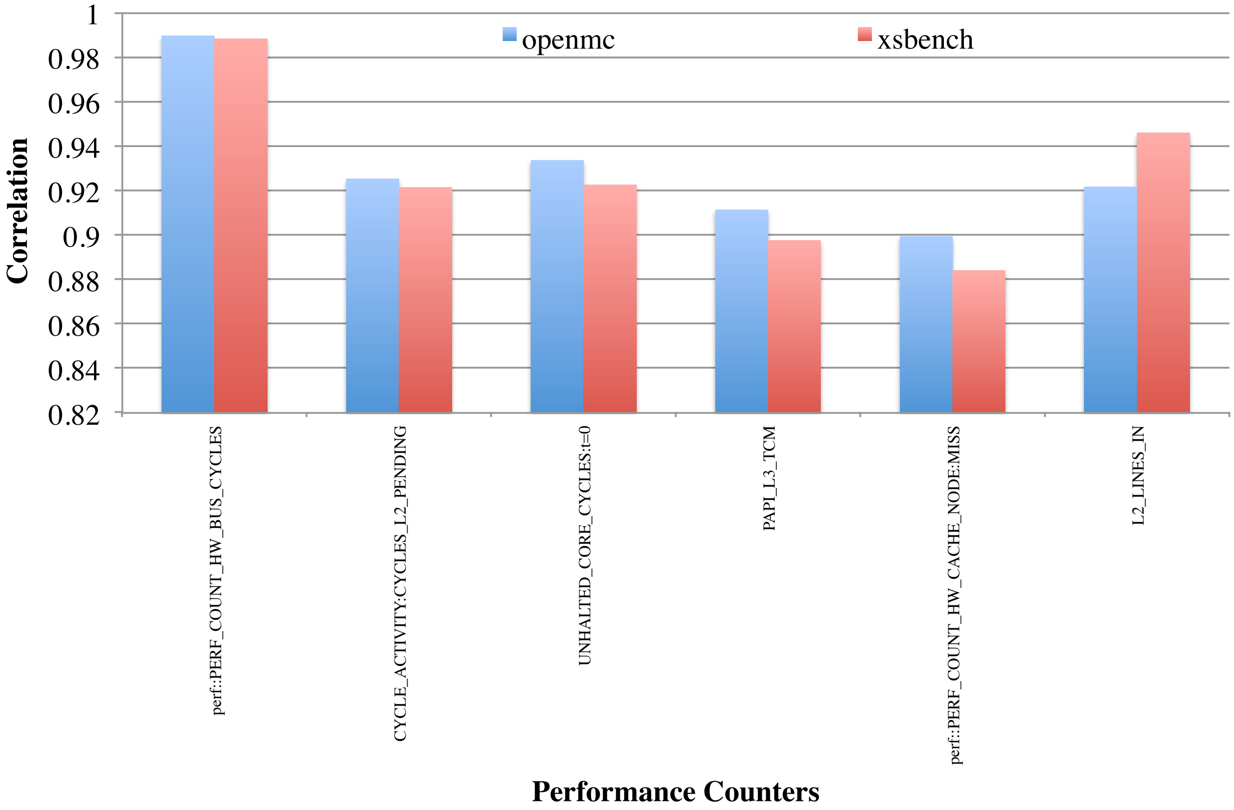

Figure 4. Ratio between correlation coefficients of *XSBench* and *OpenMC* for those variables that are also strongly correlated with their corresponding loss of efficiency.

4. CONCLUSIONS

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 the neutron transport application *OpenMC*, we have abstracted a key computational kernel that is responsible for the majority of the algorithm's runtime and implemented 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.

We have verified that *XSBench* faithfully recreates the data access patterns of the full MC application, *OpenMC*, across a broad range of hardware performance parameters. The multi-core scaling efficiency and floating point calculation rate of *XSBench* are very similar (within 5%) to *OpenMC*. Furthermore, the key portions of *XSBench*'s hardware performance profile closely match *OpenMC* – both featuring heavy contention of shared memory resources.

Thus, our analysis has shown that *XSBench* forms a good "stand-in" for the full application, *OpenMC*. Performance analysis and tuning done with *XSBench* should provide results applicable to the full MC neutron transport algorithm, while being far easier to modify, run, and interpret, making *XSBench* a convenient test bed for performance analyses on cutting edge platforms and experimental next-generation high performance computing architectures.

5. FUTURE WORK

One might reasonably conclude that 66% efficiency on 16 cores, as shown in Figure 1, is adequate scaling for *OpenMC*. However, next-generation node architectures are likely to require up to thousand-way on-node shared memory parallelism,[10][11][12][13] and thus it is crucial to ascertain the cause of the observed degradation and the implications for greater levels of scalability. Considering nodes with thousands of shared memory cores and beyond, it cannot be taken for granted that performance will continue to improve. We thus believe it important to identify to the greatest extent possible which particular system resources are being exhausted, and how quickly, so that designers of future hardware systems as well as developers of future MC particle transport applications can avoid bottlenecks. Such studies are already underway on multi-core systems[14], but further analysis is required for more exotic systems such as GPGPUs and the Xeon Phi accelerator.

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] P. K. Romano and B. Forget. "The OpenMC Monte Carlo particle transport code." *Annals of Nuclear Energy*, **51(C)**: pp. 274–281 (2013).

[2] P. K. Romano, B. Forget, and F. B. Brown. "Towards scalable parallelism in monte carlo particle transport codes using remote memory access." (pp. 17–21) (2010).

[3] "Openmc monte carlo code." `https://github.com/mit-crpg/openmc` (2014).

[4] A. R. Siegel *et al.* "Multi-core performance studies of a Monte Carlo neutron transport code." *International Journal of High Performance Computing Applications*, (pp. 1–25) (2013).

[5] "Xsbench: The monte carlo macroscopic cross section lookup benchmark." https://github.com/jtramm/XSBench (2014).

[6] J. E. Hoogenboom, W. R. Martin, and B. Petrovic. "MONTE CARLO PERFORMANCE BENCHMARK FOR DETAILED POWER DENSITY CALCULATION IN A FULL SIZE REACTOR CORE Benchmark specifications." *Ann Arbor*, **1001**: pp. 48109–42104 (2010).

[7] P. K. Romano *et al.* "Data decomposition of Monte Carlo particle transport simulations via tally servers." *JOURNAL OF COMPUTATIONAL PHYSICS*, **252(C)**: pp. 20–36 (2013).

[8] J. Leppänen. "Two practical methods for unionized energy grid construction in continuous-energy Monte Carlo neutron transport calculation." *Annals of Nuclear Energy*, **36(7)**: pp. 878–885 (2009).

[9] "Papi - performance application programming interface." http://icl.cs.utk.edu/papi/index.html (2013).

[10] S. Dosanjh *et al.* "Exascale design space exploration and co-design." *Future Generation Computer Systems*, **(0)**: pp. –. ISSN 0167-739X. URL http://dx.doi.org/http://dx.doi.org/10.1016/j.future.2013.04.018 (2013).

[11] N. Attig, P. Gibbon, and T. Lippert. "Trends in supercomputing: The european path to exascale." *Computer Physics Communications*, **182(9)**: pp. 2041 – 2046. ISSN 0010-4655. URL http://dx.doi.org/http://dx.doi.org/10.1016/j.cpc.2010.11.011 (2011).

[12] N. Rajovic *et al.* "The low power architecture approach towards exascale computing." *Journal of Computational Science*, **(0)**: pp. –. ISSN 1877-7503. URL http://dx.doi.org/http://dx.doi.org/10.1016/j.jocs.2013.01.002 (2013).

[13] C. Engelmann. "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale." *Future Generation Computer Systems*, **(0)**: pp. –. ISSN 0167-739X. URL http://dx.doi.org/http://dx.doi.org/10.1016/j.future.2013.04.014 (2013).

[14] J. Tramm and A. R. Siegel. "Memory Bottlenecks and Memory Contention in Multi-Core Monte Carlo Transport Codes." In: *Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo*. Argonne National Laboratory, Paris (2013).