

Improved cache performance in Monte Carlo transport calculations using energy banding

A. Siegel^a, K. Smith^b, K. Felker^{c,*}, P. Romano^b, B. Forget^b, P. Beckman^c

^aArgonne National Laboratory, Theory and Computing Sciences and Nuclear Engineering Division

^bMassachusetts Institute of Technology, Department of Nuclear Science and Engineering

^cArgonne National Laboratory, Theory and Computing Sciences

Abstract

We present an energy banding algorithm for Monte Carlo (MC) neutral particle transport simulations which depend on large cross section lookup tables. In MC codes, read-only cross section data tables are accessed frequently, exhibit poor locality, and are typically much too large to fit in fast memory. Thus, performance is often limited by long latencies to RAM, or by off-node communication latencies when the data footprint is very large and must be decomposed on a distributed memory machine. The proposed energy banding algorithm allows maximal temporal reuse of data in band sizes that can flexibly accommodate different architectural features. The energy banding algorithm is general and has a number of benefits compared to the traditional approach. In the present analysis we explore its potential to achieve improvements in time-to-solution on modern cache-based architectures.

Keywords: OpenMC Monte Carlo energy banding

1. Introduction

Monte Carlo (MC) neutral particle transport methods are critical for a broad range of scientific and engineering domains. Some important examples include the design, certification, and operation of nuclear reactors [1],

*Corresponding author: Argonne National Laboratory, Theory and Computing Sciences, 9700 South Cass Avenue, Building 240, 2F8, Argonne, IL 60439-4844, USA.

Email addresses: siegela@mcs.anl.gov (A. Siegel), kord@mit.edu (K. Smith), felker@mcs.anl.gov (K. Felker), romano7@mit.edu (P. Romano), bforget@mit.edu (B. Forget), beckman@mcs.anl.gov (P. Beckman)

Preprint submitted to Comput. Phys. Commun.

July 2, 2013

nuclear fusion [2], radiation shielding, weapons design, medical dosimetry [3], and cloud radiation [4]. MC methods have a long history of successfully adapting to leadership class computing architectures, including excellent scalability on distributed memory platforms [5], innovative approaches for efficient execution on vector machines [6], and more recently proof-of-principle calculations for stripped down codes on SIMD architectures [7].

The key step in all applications of MC neutron transport involves modeling the interactions of individual neutrons by randomly sampling read-only interaction probability tables. These cross section data tables represent the likelihood of all of the possible phenomena in the flight of a neutron — interaction with a particular nucleus, the resulting interaction phenomena (including absorption and scattering), scattering angle, and resulting energy. The details of these interactions are unimportant for this analysis. We merely note that the probabilities are strongly dependent on the precise energy of the neutron, and thus, as a neutron jumps around in energy from interaction to interaction, the calculation involves frequent, nearly random access to very large read-only lookup tables, something which presents significant performance challenges when executing simulations on modern CPUs.

While most neutral particle MC codes carry out the same basic set of operations, our key application driver in this analysis is the classical problem of quasistatic nuclide depletion calculations in nuclear reactor cores. This class of applications imposes very strict requirements on the most difficult computational aspects of the algorithm. Temperature dependence of the cross sections must be taken into account, and the full nuclide inventory (hundreds of nuclides in the fuel region) must be tracked across a wide range of energies. With up to 10^5 energy levels of tabulated data per temperature per nuclide, cross section lookup tables can exceed 100 GB of memory for robust reactor analysis [8, 9].

In addition to having a large memory footprint, the cross section data must be read frequently during the tracking of each individual particle—specifically, once per particle per interaction or change in material region for each nuclide in the given material region. For the classic *history method* of particle tracking, where each particle is tracked independently one by one from birth to absorption, the cross section data tables are accessed with little temporal or spatial locality. Thus, integration times are limited by the memory read latency of the system on which they are executed. For robust reactor calculations the cross section data loads can consume up to 85% of the total application time, and typical integration times of thousands of particles per

second can make robust core calculations highly impractical [10]

In this analysis we describe and test a variation of the history method that makes much more efficient reuse of fast memory to achieve improved tracking rates in the presence of large lookup tables ¹. The meaning of *fast memory* as used here is general and depends on the application context—RAM vs. off-node for large distributed data structures, or cache vs. RAM for data tables that fit on node, various levels of affinity withing a deeper NUMA hierarchy, or even RAM vs. external disk in some applications. While the implementation will differ in each case the fundamental idea is the same; the algorithm better exploits existing temporal locality inherent in the physics to maximize reuse within fast memory. The direct benefit may be improved tracking times, reductions in data movement and power usage, or a reduction in the on-node memory footprint for very large cross section lookup tables. In the current analysis our focus is demonstrating a technique for performance improvements via better use of on-node cache, both for single an multi-core implementations.

2. Algorithmic description

As mentioned in Section 1, cross section data can be as large as 100 GB for robust reactor core analysis. This implies that some decomposition strategy is required across nodes; for our proposed algorithm, memory nodes would serve as slow memory, and the local tracking node as fast memory. We call this particular implementation the *energy band memory server* algorithm and it is the subject of a companion paper [cite]. In the present work we instead consider the case where the total cross section memory resides on the local tracking node. Though this is not currently the case for many applications, our motivation in exploring this regime is the considerable recent success in cross section data compression and reduction, resulting in much smaller lookup tables with increased on-the-fly re-computation [11, 8]. These techniques in general allow some flexibility in balancing memory footprint versus FLOPS, so we consider the case where cross section data both fits on node and its size is potentially variable. The question is whether we can gain speedup in such a scenario by taking greater advantage of on-node cache.

The energy banding algorithm presented here is similar to an approach that was proposed in the 1970s designed to enable core calculations with lookup table footprints that were much larger than could be accommodated

¹Large in this context is taken to mean relative to the size of fast memory

by RAM [6]. In that case the tape drive served as a large, slow memory, and RAM served as a smaller fast memory. The idea was to load the memory in contiguous banded segments and track as many particles as possible for that band before replacing the band. As far as we know this approach was never analyzed in depth, routinely applied, or generalized to other fast/slow memory scenarios such as cache or distributed node architectures.

2.1. Classic tracking algorithm

We begin with a high-level description of the classical history method for particle tracking, and subsequently describe the energy band algorithm by comparison. Readers wishing for a more rigorous pseudo-code description of the classic algorithm are referred to [12].

Algorithm 1 Classical history algorithm

```
1: for each particle do
2:   repeat
3:     lookup material at particle position
4:     for each nuclide in local material do
5:       for each interaction type do
6:         lookup microscopic xs
7:         accumulate contribution to macro xs
8:       end for
9:     end for
10:    randomly sample interaction
11:    update particle position and energy
12:  until particle is absorbed
13: end for
```

The key operations of the history approach are described in Algorithm 1. The cross section table lookups occur at line number six. This operation requires a separate load at each iteration of the inner loop—specifically, a table lookup of the microscopic cross section for each nuclide for each interaction type at the given particle energy level. This is costly for several reasons. If we take a classic reactor core calculation as our guiding application, we note that a neutron undergoes dozens of interactions from birth to absorption in a reactor core, and the fuel regions of a reactor core contain hundreds of nuclides; thus, each individual neutron could easily require several thousand microscopic cross section lookups.

Furthermore, the size of the microscopic cross section table and unpredictable access pattern yield little cache reuse and is unfriendly to data prefetchers. A single nuclide typically entails close to 100,000 tabulated energy levels for a single interaction type at a single temperature value. Without data reduction, given that a dozen reaction types, hundreds of nuclides, and approximately fifty temperatures are required for an accurate calculation, the tabulated cross section data can exceed 100 GB of memory. Even with data reduction, where an order of magnitude or more reduction in memory footprint is possible, there is still no indication that reducing the data to even typically L3 cache sizes is even close to possible. Furthermore, though neutrons tend to travel from higher to lower energies (thermal upscattering occurs in light water reactors), the precise cross section values jump around from interaction to interaction and are unpredictable in their details.

2.2. Energy banding algorithm

Our proposed energy banding history algorithm constitutes a small modification of Algorithm 1. The idea is to exploit knowledge of the *gross path* through energy space of each individual neutron, independent of the details. Neutrons are born at very high energies (~ 2 MeV) and slow down through series of scattering interactions through thermal regime (0.025 eV). In a traditional light water cooled nuclear reactor (LWR), for example, this path is for the most part uni-directional— i.e. particles move from higher to lower energies (some upscattering may occur in the thermal energy regime).

The energy banding algorithm first partitions the cross section data into n energy bands E_n, E_{n-1}, \dots, E_1 . For simplicity we may assume that the bands are equally divided in terms of memory footprint. If the energy band size is chosen so as to fit in fast memory, then we may obtain significant fast memory reuse by adding an outer loop over bands. Specifically, each particle is then tracked either until absorption or until the particle leaves the energy band. All particles in a given band are then processed before the next band is loaded into fast memory.

With very large particle numbers typically required for adequate statistical convergence, the chance of reusing cache lines before eviction is much higher, and we may expect significant speedup compared to the classic algorithm. The energy banding algorithm is described in Algorithm 2 below. Note that an additional outer loop (not shown) is required in the presence of upscatter.

Algorithm 2 Energy band algorithm

```
1: partition  $[E_0, E_{max}]$  into  $n$  energy bands
2: for each energy band do
3:   for each particle in band do
4:     repeat
5:       lookup material at particle position
6:       for each nuclide in local material do
7:         for each interaction type do
8:           lookup microscopic xs
9:           accumulate contribution to macro xs
10:        end for
11:       end for
12:       randomly sample interaction
13:       update particle position and energy
14:     until particle is absorbed or particle leaves energy band
15:   end for
16: end for
```

3. Numerical experiments

While the energy banding algorithm has a number of potential advantages, our interest in the present context is as an efficient on-node algorithm for current and next generation node design. This includes both obtaining speedup on traditional single core cache-based architectures but also deeper issues such as limiting data movement (and thus power) and assessing on-node scalability for both multicore and potentially many-core architectures.

3.1. Proxy application

To carry out our analysis we develop a simplified MC proxy application that mimics the workload of the full algorithm necessary to reveal key performance characteristics. The proxy application uses homogenized data from our full physics application, OpenMC [13], to grossly represent the key features of the full tracking algorithm. Specifically, the proxy application takes as input the number of particles, the size of the cross section data array, the number of energy bands, and the number of (shared memory) OpenMP threads. It produces a tracking rate by integrating each particle through a series of randomly sampled interactions in a homogeneous material. At each interaction particle absorption is determined probabilistically by a gross ab-

sorption rate probability, and energy group scattering probabilities are determined by an $n \times n$ user-input scattering matrix, which for the tests executed here is derived from execution of our full physics solver OpenMC on a classic reactor benchmark calculation [14]. At each interaction point the cross section data buffer loads m cross section data values from randomly sampled locations within the energy band. To properly model the application the value of m must be chosen based on the number of nuclides in the target simulation. In our tests we showed, not surprisingly, that there is minimal discernible effect on our results as this number is varied.

The proxy application then proceeds as described in Algorithm 2 above. Each band is stored contiguously in memory and all particles are integrated until they are absorbed or leave the energy band. Since tens of thousands of particles are integrated within each band and the sampling is random within a band, if the band fits in fast memory we expect most of the cross section reads to be cache hits, and the footprint of the cache should be clear with varying band size. Parallelization was carried out using the OpenMP *parallel for* construct on the particle loop. While this in principle could impose load imbalances as each band is forced to proceed in lockstep, in practice there are enough particles per core to ensure that on average integration times do not vary significantly. Furthermore, this is the desired approach in multicore for higher levels of cache that are typically shared among cores— if bands were not processed in lockstep cache overwriting between bands would severely hamper performance.

3.2. Results

Figure 1 shows performance results for relative tracking time as a function of cross section data table size. We normalize all timing figures by the maximum value for the single threaded case, so that all data can be interpreted as relative performance loss. The performance tests were carried out on dual Intel Xeon E5430 2.66 GHz processor, each with 4 cores, for a total of 8 shared memory cores. The four tiled plots represent respectively 1,2,4, and 8 core tests, and within each test five levels of energy banding are shown.

Our first question is what level of speedup can be achieved by using the energy banding algorithm with bands that fit in cache. In this analysis, the cache of interest is the 12 MB shared L3 cache. As discussed previously, though a number of ideas are being pursued to dramatically reduce on-node memory footprints for cross section data tables, it is unrealistic to suppose that the 32 kB and 256 kB L1 and L2 caches could be used without massive numbers of energy bands. When the bands are too small relative to the total

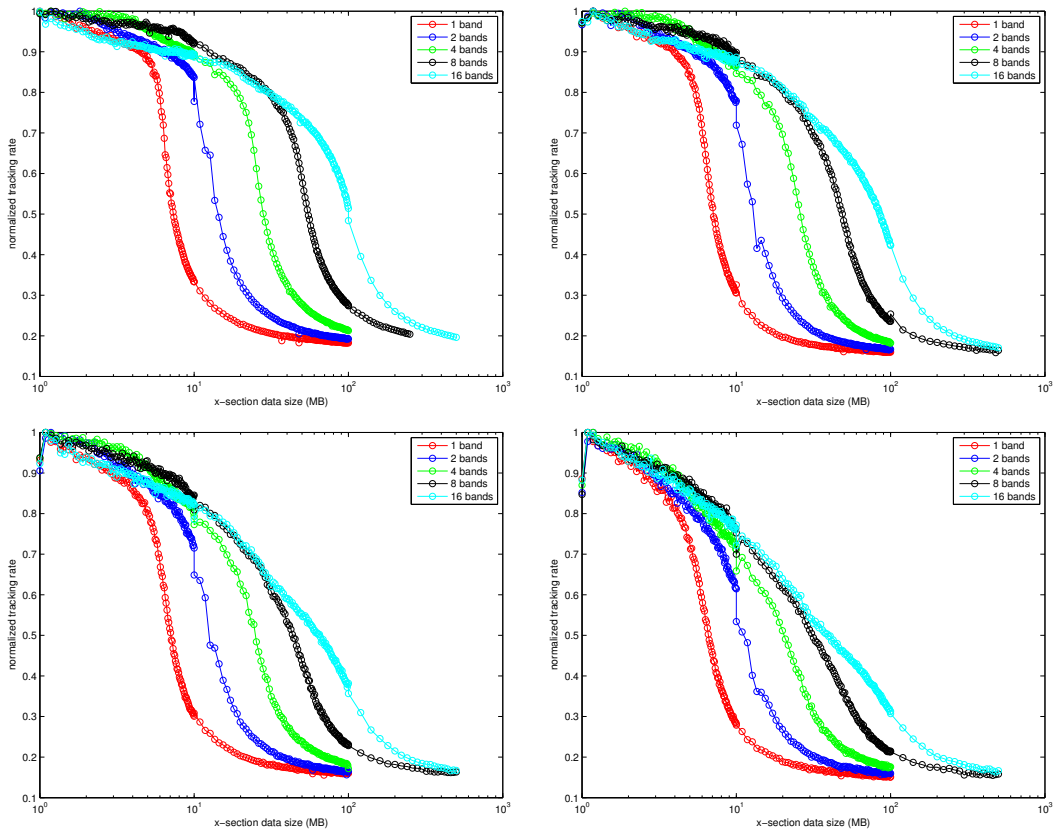


Fig. 1. Starting in the top left tile and proceeding clockwise, the plots represent the performance of the on-node EBMS algorithm for 1,2,8,4 OpenMP threads. The performance timings were normalized to the maximum tracking rate of the single thread case.

cross section data size, relatively few particles will have interactions within a band, and the advantage of locality will be severely limited. Given this observation we have restricted our analysis to the more reasonably sized L3 cache.

Figure 1a illustrates the significant advantage of the banding when the band sizes are judiciously chosen to be slightly smaller than the L3 cache size. We interpret the performance results as follows. First, observing the single band curve in the single threaded case (red line in upper left plot), it is clear that as cross section data grows to be larger than the L3 cache size, particle tracking rates reduce dramatically, in this case by a factor of five slower than the peak observed value. This speedup is significant and would lead us to seek cross section data reduction near L3 cache sizes when possible. An alternative that allows more flexibility (and is likely more feasible) is to use our banding algorithm to enable similar speedups for larger data footprints. Observing the 2,4,8, and 16 band results, we see that it is possible to reach near-peak tracking rates for proportionately larger cross section data footprints. Our tests further reveal that the small erosion relative to peak performance is explained largely by the smaller number of interactions per band as band sizes decrease. This leads to less cache reuse and ultimately puts a limit on the number of bands that can be used while still obtaining significant performance improvements.

A related issue of importance is the on-node scalability of the banded algorithm in the presence of a complex, shared memory hierarchy. A cache coherent distributed L1 cache and shared L2 and L3 caches present a significant challenge for the scalability of particle tracking algorithms [12]. In the development of modern computational physics applications, a key question for new algorithms is not just their single core performance characteristics but also their performance and scalability on multicore, shared memory nodes. This is particularly important when the relevant algorithm is highly sensitive to the properties of the on-node memory hierarchy. In the present case our hope is that multicore implementations of the banding algorithm show similar performance and reasonable scaling properties.

This question is explored in Figure 1b, 1c, and 1d, where 2, 4, and 8 core experiments are carried out for the same cases as Figure 1a. On-node parallelization was implemented with the OpenMP library by threading the particle loop within each band via a *parallel for* construct before line 3 in Algorithm 2. This approach has been referred to as *coarse-grained* threading in the MC community and has been shown to give reasonably good performance

for large particle counts [12].

Results in Figures 1b-d are still normalized by the maximum single threaded tracking rate, so in each case we clearly see qualitatively nearly ideal speedup with thread count. More significantly for the present analysis, we can observe whether the banding algorithm continues to provide benefit when executed with on-node parallelism. The results are mixed in this regard. Speedup is still significant for larger band sizes, but the drop-off from peak tracking rates is more abrupt and the range over which the banding algorithm gives significant benefit is less broad, particularly for the eight-core experiments. Nonetheless we still observe speedups of over a factor of three for a fairly significant range of energy sizes, indicating that the algorithm is still likely beneficial for relatively modest core sizes. Performing efficiently on many core nodes involves many deep questions that are beyond the scope of the present analysis.

4. Conclusion

We have demonstrated a flexible algorithm which exploits basic properties of neutron physics to make more efficient reuse of fast memory in neutral particle MC transport simulations. The algorithm can to some extent adjust to a range of cross section memory footprint sizes so that the user can optimally tune it for their particular application. In the present analysis we considered the application of this algorithm in the case that cross section data could be reduced to a reasonable multiple of L3 cache data size. In this case L3 and RAM then served as fast/slow memory, respectively.

We carried out proof-of-principle tests of our banding algorithm on a simplified proxy application that models the key aspects of a full neutron transport code, using as input scattering and absorption statistics from the OpenMC application code. Our results indicated that significant speedup (up to a factor of five) is attainable when memory bands can be forced into L3 cache, so long as an adequate number of particles is used to force significant cache reuse. Furthermore, this effect was observed to persist in the presence of on-node parallelism, though the benefits were reduced to a moderate degree in our largest (eight core) experiments.

Acknowledgment

This work was supported by the U. S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357.

References

- [1] Y. Azmy, E. Sartori, J. Spanier, Monte Carlo methods, in: Nuclear Computational Science, Springer Netherlands, 2010, pp. 117–165.
- [2] D. Heifetz, D. Post, M. Petravic, J. Weisheit, G. Bateman, A Monte-Carlo model of neutral-particle transport in diverted plasmas, *J. Comput. Phys.* 46 (2) (1982) 309 – 327.
- [3] D. W. O. Rogers, Fifty years of Monte Carlo simulations for medical physics, *Phys. Med. Biol.* 51 (13) (2006) R287.
- [4] W. O’Hirok, C. Gautier, A three-dimensional radiative transfer model to investigate the solar radiation within a cloudy atmosphere, Part I: spatial effects, *J. Atmos. Sci.* 55 (12) (1998) 2162–2179.
- [5] P. K. Romano, B. Forget, F. Brown, Towards scalable parallelism in monte carlo particle transport codes using remote memory access, *Prog. Nucl. Sci. Technol.* 2 (2011) 670–675.
- [6] F. B. Brown, W. R. Martin, Monte Carlo methods for radiation transport analysis on vector computers, *Progress in Nuclear Energy* 14 (3) (1984) 269–299.
- [7] F. A. van Heerden, A coarse grained particle transport solver designed specifically for graphics processing units, *Transport Theory and Statistical Physics* 41 (1-2) (2012) 80–100. doi:10.1080/00411450.2012.671215.
- [8] W. R. Martin, S. Wilderman, F. B. Brown, G. Yesilyurt, Implementation of on-the-fly doppler broadening in mcnp, in: International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, Idaho, 2013.
- [9] A. Siegel, K. Smith, P. Romano, B. Forget, K. Felker, The effect of load imbalances on the performance of Monte Carlo algorithms in LWR analysis, *J. Comput. Phys.* 235 (2013) 901–911. doi:10.1016/j.jcp.2012.06.012.
- [10] D. J. Kelly, T. M. Sutton, T. Trumbull, P. S. Dobreff, MC21 Monte Carlo analysis of the Hoogenboom–Martin full-core PWR benchmark problem, in: PHYSOR – Advances in Reactor Physics to Power the Nuclear Renaissance, 2010.
- [11] T. Viitanen, J. Leppänen, Explicit treatment of thermal motion in continuous-energy monte carlo tracking routines, *Nucl. Sci. Eng.* 171 (2012) 165–173.
- [12] A. Siegel, K. Smith, P. Romano, B. Forget, K. Felker, Multi-core performance studies of a monte carlo neutron transport code, *International Journal of High Performance Computing Applications*.
- [13] P. K. Romano, B. Forget, The OpenMC Monte Carlo particle transport code, *Ann. Nucl. Energy* 51 (2013) 274–281. doi:10.1016/j.anucene.2012.06.040.
- [14] J. E. Hoogenboom, W. R. Martin, B. Petrovic, The Monte Carlo performance benchmark test - aims, specifications and first results, in: International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Rio de Janeiro, Brazil, 2011.

Government license

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. 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.