USING FRACTIONAL CASCADING TO ACCELERATE CROSS SECTION LOOKUPS IN MONTE CARLO NEUTRON TRANSPORT CALCULATIONS

Amanda L. Lund[1], Andrew R. Siegel[1], Benoit Forget[2], Colin Josey[2], and Paul K. Romano[3]
[1] Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave, Building 240, Lemont, IL 60439, alund@anl.gov; siegela@mcs.anl.gov
[2] Department of Nuclear Science and Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, 24-107, Cambridge, MA 02139, bforget@mit.edu; cjosey@mit.edu
[3]Bechtel Marine Propulsion Corporation, Knolls Atomic Power Laboratory, P.O. Box 1072, Schenectady, NY 12301, United States, paul.romano@unnpp.gov

ABSTRACT

We describe and test a technique for carrying out energy grid searches in continuous-energy Monte Carlo (MC) neutron transport calculations that represents an optimal compromise between grid search performance and memory footprint. The method, based on the fractional cascading technique and referred to as the *cascade grid*, is tested within the OpenMC Monte Carlo code, and performance results comparing the method with existing approaches are presented for the Hoogenboom-Martin reactor benchmark. The cascade grid achieves significant speedups in calculation rate with negligible initialization overhead while not increasing the memory footprint by more than 2×.

Key Words: OpenMC, Monte Carlo, fractional cascading, cross section

1 INTRODUCTION

Robust depletion analysis of large power reactors is for all practical purposes still beyond the reach of Monte Carlo (MC) codes on present-day systems. For example, in [1] the authors estimate that a single state point calculation of a realistic reactor geometry requires many thousands of core hours on existing architectures. While the performance of MC codes in general is typically associated with random number generation and the cost of complex nested branching logic, recent research indicates that the principle performance bottleneck for problems with large nuclide inventories lies elsewhere – specifically, a small set of operations that include the search and retrieval of cross section values at the point of each neutron interaction [2].

For a robust light-water-reactor depletion calculation hundreds of nuclides need to be tracked within the fuel regions; thus, calculating the macroscopic cross section at each collision point requires essentially random lookups from cross section data tables that are much larger than last-level cache. Tramm et al. [2], for example, shows that in the OpenMC code a modified form of the Hoogenboom-Martin (H-M) reactor benchmark [3] with 362 nuclides devotes 85% of its time retrieving cross section data. Related studies confirm this result and go further in elucidating the underlying source of the bottleneck in terms of the exhaustion of hardware resources, both for single and multicore [4] analyses.

In fact, when analyzing the problem more deeply it was shown in [2] that the cross section lookup process is composed of two distinct parts, one of which is explicit in the algorithm – the energy grid index search for each nuclide – and another which is implicitly part of the computer architecture – the memory latency associated with out-of-cache memory references, including various forms of contention on multicore systems. The energy grid search itself consists of finding the bounding cross section values for the incident neutron energy for each of several hundred nuclide grids. There are several techniques in practice for doing this, balancing performance with memory footprint. These are discussed in further detail in section 2. The most straightforward approach, what we refer to as a *nuclide grid*, involves a simple binary search on each nuclide grid with complexity $k \log n$, where n is the average number of energy points for a given nuclide and k is the number of nuclides . The memory latency bottleneck is more complex. Tramm et al. [2] showed that 65% of all cross section references resulted in last level cache misses. Prefetchers and caching algorithms work poorly given the quasi-random nature of incident neutron energy levels. On modern architectures out of cache memory references incur several hundred clock cycles of latency, which for certain applications was shown to be an even larger source of performance degradation than the search itself.

Several investigators have recently proposed new techniques to improve performance of both aspects of the cross section lookups mentioned above. Leppänen [5] introduces both a unionized and universal energy grid, showing significant performance improvements at the cost of greatly increased memory footprint. Brown [6] proposes a new hash-based energy lookup algorithm that reduces the length of the grid index search by providing search bounds for each nuclide, producing speedups with little increase in memory footprint. In [7] a classic “energy banding” algorithm is reinterpreted to reduce lookup latency by maximizing the chance of cache reuse. In [8], the authors leverage the superior bandwidth of GPUs to offload large numbers of parallel cross section lookups to effectively mask latency.

In the present work our focus is optimization of the grid index search problem as one key step in improving time to solution and memory footprint in MC codes. To accomplish this we apply the *fractional cascade* (FC) technique [9] to the nuclide grid indices, resulting in a method that aims to provide an optimal compromise between grid search performance and memory footprint. The goal is to achieve memory usage close to that of the nuclide grid but with runtime performance comparable to the unionized grid. The FC method is programmed in the OpenMC code [10], and performance results are compared with several existing approaches for the H-M benchmark. Overall the FC approach shows comparable or superior runtime performance, negligible initialization overhead, and a memory footprint always less than twice the nuclide grid approach.[*]

[*] We consider the universal grid impractically large for robust depletion calculations of large power reactors.

2 BACKGROUND

In continuous-energy MC codes, the macroscopic cross section must be recalculated each time a particle changes energy (through a collision) or travels into another material. These computations require the storage of microscopic cross section values for each reaction type for every nuclide in the model over the range of all possible particle energies. The cross section tables are formulated to be dense enough such that linear interpolation between the bounding values for any particle energy will produce a result within a specified tolerance. In order to ensure adequate precision of the interpolated values, the number of the data points and the distribution of energies is highly dependent on the nuclide. In fact, the number of energy values can differ by over two orders of magnitude between elements depending on the complexity of their cross section functions. This disparity between the size and distribution of the data necessitates a separate energy and cross section grid for each nuclide.

The most straightforward approach to storing the data and performing the cross section lookups is referred to as the nuclide grid method. Every nuclide has its own distinct grid of energy values as well as a grid of the microscopic cross section values corresponding to each energy. Since the distribution of the energy values is unique to the nuclide, every time the microscopic cross section for a particular nuclide must be determined a binary search is performed on that nuclide's energy grid. Though the binary search itself takes only $O(\log n)$ time for an energy grid of length n, the search must be repeated for each nuclide in a material each time the macroscopic cross section is computed. The space required to store the grid structure is minimal compared to other schemes, but the repeated searches have a noticeable detrimental effect on performance, especially when the material is composed of a large number of nuclides (for example, in the case of fuel).

A method originally described by Leppänen [5] and currently implemented in OpenMC known as the unionized grid sacrifices memory in order to improve performance by precomputing some information. The unionized grid decreases the time spent doing cross section lookups by reducing the number of grid index searches in any material to a single binary search. To accomplish this, a supplementary energy grid structure is created on top of the nuclide grid. This structure consists of a single comprehensive energy grid constructed from the union of the energy values of each of the k nuclide's energy grids as well as a list of k integers, referred to as pointers, associated with each energy in the unionized grid. The pointer associated with energy E on the unionized energy grid and with nuclide i identifies the index that would be returned in a search for E in i's nuclide energy grid. This new data structure accelerates the grid index search by requiring only a single binary search to locate the particle's energy on the unionized grid. The location of the energy on the nuclide grids and the related cross section values are then determined through a series of indirections using the corresponding pointers. Though a significant performance improvement can be achieved using this method, it is done at the cost of increasing the memory footprint of the energy and cross section tables by approximately an order of magnitude.

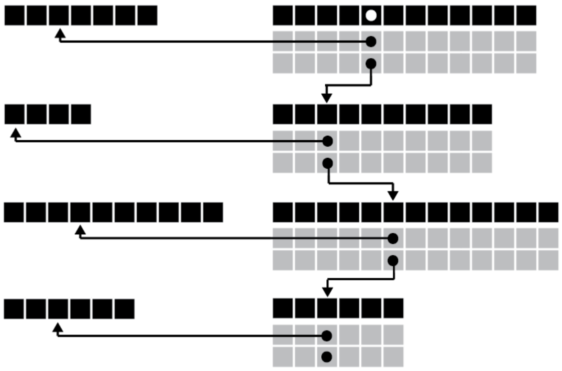

Figure 1. A representation of the cascade grid structure and lookup procedure for a model with 4 nuclides, with the original nuclide energy grids shown on the left and the augmented energy grids and pointers shown on the right. A single binary search is performed on the top augmented grid. The first pointer indicates the location of the energy on the nuclide grid, and the second pointer indicates the location of the energy on the next augmented grid. The arrows show the process of following the pointers and cascading down the grids to find the location of the energy on each nuclide grid.

3 THE CASCADE GRID AND LOOKUP ALGORITHM

In an effort to preserve the smaller space requirement of the nuclide grid structure while achieving a faster query time, the FC technique is employed to construct what we will refer to as the *cascade grid* and to accelerate the cross section lookups without any loss in data or accuracy. Like the unionized grid, the cascade grid method boosts performance by supplementing the nuclide grid structure with additional energy arrays and precomputed information about the location of energy values, but it does so while increasing the memory footprint by no more than $2\times$.

The cascade grid structure consists of the original k nuclide energy grids, a new set of k augmented energy grids, and a pointer pair associated with each value in the augmented grids. To build the augmented grids, each of the original energy grids is expanded with values from the grid immediately following it in order to correlate the two so that a lookup in one facilitates a lookup in the other. The pair of pointers accompanying an augmented grid energy value E provide information about the location of E on other energy grids: the first pointer is the index of E on the original nuclide grid, and the second pointer is the approximate index of E on the following augmented grid. The structure is designed to maintain a correspondence both between the adjacent pairs of augmented grids and between each augmented grid and the original grid from which it was constructed.

More precisely, let $\{M_1, M_2, \ldots, M_k\}$ denote the set of augmented energy grids in a model with k total nuclides and with nuclide energy grids $\{L_1, L_2, \ldots, L_k\}$. Furthermore, let each energy $E \in M_i$ have two pointers, p_1 and p_2, associated with it. In OpenMC, the cascade grid data structure is implemented in the following manner:

1. The final augmented energy grid M_k is the same as the final energy grid L_k
2. The augmented energy grid M_i is a sorted list containing every element in L_i and every second element in M_{i+1}
3. The pointer p_1 associated with energy E in M_i is the index of E in L_i
4. The pointer p_2 associated with energy E in M_i is the approximate index of E in M_{i+1}. Specifically, it is either the index of E in M_{i+1} or one more than the index of E in M_{i+1}

From 2, the size of an augmented list M_i can be shown by induction to be

$$|M_i| = |L_i| + \frac{1}{2}|L_{i+1}| + \ldots + \frac{1}{2^j}|L_{i+j}| + \ldots + \frac{1}{2^{k-i}}|L_k|. \tag{1}$$

The upper bound on the total size of the data structure can then be shown to be twice the size of the original set of lists:

$$\sum_{i=1}^{k} |M_i| = |L_1| + \left(1 + \frac{1}{2}\right)|L_2| + \ldots + \left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{k-1}}\right)|L_k| \leq 2\sum_{i=1}^{k} |L_i| \tag{2}$$

Figure 1 illustrates the cascade grid structure and lookup process. To locate the position of an energy E in each of the nuclide energy grids, a binary search is performed on only the first augmented grid M_1. The pointer p_1 associated with the returned index then identifies the position that would be returned when searching for E in the nuclide grid L_1, and the pointer p_2 identifies the approximate position that would be returned when searching for E in the following augmented grid M_2. To determine the true location of E in M_2, a single comparison is made between the value at p_2 and the value at $p_2 - 1$. This process of following the pointers is repeated down to the last augmented grid M_k. The procedure for calculating the macroscopic cross section of a material using the cascading energy grid structure where the given material region is composed of m nuclides and each nuclide i in the material has atomic density ρ_i is presented in Algorithm 1.

Algorithm 1 Cascade Grid Macroscopic Cross Section Lookup

```
for $i \leftarrow 1, m$ do                          ▷ for each nuclide in material
    if i = 1 then                                   ▷ if on first nuclide
        $j \leftarrow$ binary search$(M_i, E_p)$    ▷ find index of energy using binary search
    else                                            ▷ if not on first nuclide
        $j \leftarrow p_2$                          ▷ find index of energy using pointer
    end if
    $p_1,\ p_2 \leftarrow M_i,\ j$                  ▷ lookup pointers associated with energy
    $\sigma \leftarrow i,\ p_1$                     ▷ lookup microscopic cross section
    $\Sigma \leftarrow \Sigma + \rho_i\sigma$       ▷ accumulate macroscopic cross section
end for
```

In problems where the number of nuclides in a material varies significantly (e.g., in a problem that contains both fuel, made up of hundreds of nuclides, and water, consisting of only two), there would be a penalty incurred for cascading through the full set of nuclides to retrieve the values for any material composed of only a few nuclides. In OpenMC, the cascade grid is implemented such that the binary search is performed on the augmented array belonging to the first nuclide in the material (rather than the first nuclide in the problem) and the cascade process terminates at the last nuclide in the material. The cascade structure is ordered by nuclide in a way that reduces the depth needed to cascade down the grids to retrieve the values for each material. As a result, overhead from unnecessary null operations is avoided.

4 NUMERICAL EXPERIMENTS

To test the performance of the cascade grid algorithm, OpenMC was modified to provide the runtime option to store the nuclear data in cascade grid format. A series of numerical experiments were then carried out to measure the performance and memory footprint of the cascade grid compared to the nuclide grid and the unionized grid. Two benchmark problems were used – the popular Hoogenboom-Martin performance benchmark, referred to as *Small H-M* in the tables, with 68 total nuclides, and a modified version of Hoogenboom-Martin which uses a full nuclide inventory in the fuel region, referred to as *Large H-M*, with 362 total nuclides. For the Large H-M Tramm et al. [2] observe that 85–90% of the computation time was spent in the cross section lookup loop, which then provides a good use case for expected performance improvements of the cascade grid approach. All experiments were carried out on both a PC node consisting of two Intel Xeon E5-2650 octo-core CPUs and a single node of the IBM Blue Gene/Q (BG/Q) supercomputer *Mira* consisting of 16 physical CPUs. We commonly employ these systems for production computing, and as they represent two vastly different strategies in architecture design it is useful to test and compare our conclusions between them.

Table I. Memory footprint of the three energy grid methods.

	Small H-M	Large H-M
Nuclide Grid	52 MB	157 MB
Cascade Grid	94 MB	299 MB
Unionized Grid	408 MB	5637 MB

To gain a comprehensive view of the tradeoffs and limitations of each algorithm, we focused on comparing four key quantities: memory footprint, grid creation startup cost, cumulative macroscopic cross section lookup time, and resulting particle tracking rate. Table I presents the memory requirement of each of the three grid structures for both the Large H-M and Small H-M benchmarks. The unionized grid uses 8× more memory than the nuclide grid for the Small H-M, and scales poorly as the number of nuclides in the problem increases, requiring 36× more memory (5.6 GB total) than nuclide grid structure for the Large H-M. The space requirement of the cascade grid structure, on the other hand, is never more than twice that of the nuclide grid and even for the Large H-M is only a few hundred MB.

Results for grid creation startup cost, total time spent carrying out the macroscopic cross section lookups, and particle tracking rate are presented in Table II for the two different compute systems mentioned previously. The speedup and relative timings observed between the three grid methods

Table II. Timing results for the three energy grid methods.

	Intel Xeon E5-2650			IBM Blue Gene/Q Vesta		
	Creation Time	Σ Lookup Time	Calculation Rate	Creation Time	Σ Lookup Time	Calculation Rate
Small H-M						
Nuclide Grid	0.00 s	41.4 s	3780 n/s	0.0 s	315.3 s	440 n/s
Cascade Grid	0.03 s	24.1 s	5610 n/s	0.1 s	200.3 s	588 n/s
Unionized Grid	1.72 s	30.2 s	4780 n/s	13.1 s	207.6 s	576 n/s
Large H-M						
Nuclide Grid	0.00 s	437.3 s	458 n/s	0.0 s	2265.5 s	83 n/s
Cascade Grid	0.09 s	371.1 s	525 n/s	0.3 s	1748.9 s	105 n/s
Unionized Grid	36.84 s	329.3 s	640 n/s	156.2 s	1275.0 s	140 n/s

are consistent between the two systems. The grid creation time for the cascade method is negligible, never exceeding a few hundred milliseconds on either system for either problem size. The unionized grid, on the other hand, takes a significant amount of time to set up, from seconds to several minutes on BG/Q running the Large H-M. This startup cost can amount to a non-trivial proportion of the total runtime depending on the problem, but is typically most significant when doing frequent, high-turnaround test cases.

The performance benefit of the cascade and unionized grids is evident when comparing the time spent doing the macroscopic cross section lookups. Both techniques are able to greatly reduce the computational time spent on the energy search and cross section retrieval, which translates into an improved calculation rate. The unionized grid method consistently achieves a higher particle tracking rate (measured in neutrons/second) than the nuclide grid. However, while requiring only a single binary search, the unionized grid has greater hidden memory costs due to indirection, and thus in practice some erosion of the expected performance benefit given strictly the reduced number of operations is observed. The cascade grid method also generates considerable speedups in calculation rate for each benchmark and compute system. Compared against the nuclide grid method, the cascade grid attains tracking rate speedups of 1.2-$1.5\times$. Additionally, it demonstrates comparable or superior performance to that of the unionized grid. When run on the Small H-M benchmark on either system, it performs better than the unionized grid method, though on the Large H-M benchmark its calculation rate is slightly lower. However, it possesses the added benefits of small memory usage and trivial startup cost.

5 CONCLUSIONS

Applying the FC technique to the energy grid index search problem in the OpenMC particle transport code produces considerable speedups in the particle tracking rate with trivial initialization overhead and little increase in memory usage. The performance is consistently superior to that of the nuclide grid method and comparable to or better than that of the unionized grid method depending on the model. Unlike the unionized grid method, the startup cost for the cascade grid is negligible. Additionally, the memory footprint of the cascade grid never exceeds twice that of the nuclide grid, requiring at most a few hundred MB for the Large H-M benchmark compared to the several GB required of the unionized grid method. The cascade grid presents a suitable compromise between performance and memory footprint for continuous-energy Monte Carlo transport calculations.

6 REFERENCES

[1] K. Smith, "Reactor Core Methods," *Proceedings of M&C 2003 International Conference*, Gatlinburg, Tennessee, USA, April 6–10, 2003.

[2] J. R. Tramm and A. R. Siegel, "Memory Bottlenecks and Memory Contention in Multi-Core Monte Carlo Transport Codes," *Proceedings of SNA+MC 2013*, Paris, France, Oct 27–31, 2013.

[3] J. E. Hoogenboom, W. R. Martin, and B. Petrovic, "The Monte Carlo Performance Benchmark Test – Aims, Specifications, and First Results," *Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Rio de Janeiro, Brazil, May 8–12, 2011.

[4] A. R. Siegel, K. Smith, P. K. Romano, B. Forget, and K. G. Felker, "Multi-core performance studies of a Monte Carlo neutron transport code," *The International Journal of High Performance Computing Applications*, **28**, pp. 87–96 (2014).

[5] 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).

[6] F. B. Brown, "New Hash-Based Energy Lookup Algorithm for Monte Carlo Codes," *Trans. Am. Nucl. Soc.*, **111**, *1*, pp. 659–662 (2014).

[7] A. Siegel, K. Smith, K. Felker, P. Romano, and B. Forget, "Improved cache performance in Monte Carlo transport calculations using energy banding," *Comput. Phys. Commun.*, **185**, pp. 1195–1199 (2014).

[8] A. R. Siegel, J. Tramm, T. Scudiero, and P. K. Romano, "A strategy for accelerating Monte Carlo criticality calculations using GPUs," (2014), Manuscript submitted for publication.

[9] B. Chazelle and L. J. Guibas, "Fractional cascading: I. A data structuring technique," *Algorithmica*, **1**, *1-4*, pp. 133–162 (1986).

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