Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)
La Cité des Sciences et de l'Industrie, Paris, France, October 27–31, 2013

The energy band memory server algorithm for parallel Monte Carlo transport calculations

Kyle G. Felker[1], Andrew R. Siegel[2], Kord S. Smith[3], Paul K. Romano[3], and Benoit Forget[3]

[1]*Argonne National Laboratory, MCS Division, 9700 S. Cass Ave, Building 240, Argonne, IL 60439*
[2]*Argonne National Laboratory, MCS and NE Divisions, 9700 S. Cass Ave, Building 240, Argonne, IL 60439*
[3]*Massachusetts Institute of Technology, Department of NSE, 77 Massachusetts Ave, Cambridge, MA 02139*

An algorithm is developed to significantly reduce the on-node footprint of cross section memory in Monte Carlo particle tracking algorithms. The classic method of per-node replication of cross section data is replaced by a memory server model, in which the read-only lookup tables reside on a remote set of disjoint processors. The main particle tracking algorithm is then modified in such a way as to enable efficient use of the remotely stored data in the particle tracking algorithm. Results of a prototype code on a Blue Gene/Q installation reveal that the penalty for remote storage is reasonable in the context of time scales for real-world applications, thus yielding a path forward for a broad range of applications that are memory bound using current techniques.

KEYWORDS: Monte Carlo, memory server, cross section, energy band

I. Introduction

Typical applications of continuous energy Monte Carlo neutron transport codes require frequent table lookups of large cross section data libraries. This is particularly true of nuclear reactor core analyses, for which a typical full cycle depletion calculation requires tracking inventories of several hundred nuclides within the nuclear fuel regions. When temperature dependence between 300 and 3000 K is included, cross section data lookup tables can easily exceed 100 GB of memory.(1) Since the cross section data must be read potentially many times during the tracking of an individual particle (once per particle per interaction or change in material region), and the data is accessed more or less randomly from interaction to interaction, the standard approach to achieving good performance involves storing the entire cross section table in memory for the duration of the algorithm. For common distributed memory parallel MC implementations where each processor is assigned an equal subset of particles to track, the cross section libraries are replicated on each distributed processing node.

Since the cross section data memory in general far exceeds per node limits, this brute force strategy is only feasible when the application is idealized in order to significantly reduce the volume of cross section data. For classical nuclear reactor core physics calculations, the problem is typically simplified by significantly reducing the actual number of nuclides and material regions required for robust core analysis, yielding table sizes that fit in on-node memory (approximately 10 GB is typical). In addition to falling short of a realistic core simulation, the heavy memory burden of cross section data also constrains the size of other important data structures such as interaction tallies and particle storage, limiting the detail of the largest simulation that can be carried out.(2)

For large parallel simulations on distributed memory platforms, there is more than enough *aggregate* memory to accommodate cross section tables for a robust reactor analysis. The key question, though, is how to decompose storage without an enormous cost to time-to-solution. An algorithm with reduced memory footprint but much longer integration times is unlikely to be of practical value. Accordingly, a good metric for success of a data decomposition algorithm is to enable sufficient aggregate storage while limiting the impact on performance to a reasonable fraction of overall simulation time. In this analysis we develop such an algorithm based on basic principles of overall locality in per-neutron access patterns of cross section data and implemented using a set of dedicated *tracking processors* to access data on a remote, disjoint set of *memory server* processors.

Some have suggested that "bringing the data to the particle" is a more natural and scalable approach on the trajectory to exascale computing than "moving the particle to the data" (e.g. physical space domain decomposition).(3) Moreover, the key underlying concepts of *energy bands* or *supergroups*, which are the foundation of the present algorithm, have their roots in early work done in related but distinct contexts, for example when carrying out Monte Carlo in serial but using external tape storage, or in devising vectorized algorithms for vector-based supercomputers of the 1980's.(4)(5) Romano et al. recently experimented with decomposing cross section data using remote memory access operations and, more promisingly, dividing the tally data into a client-server model.(6)(7) Similar client-server models have been used successfully in quantum Monte Carlo calculations.(8) This analysis represents one specific realization of these general ideas with the goal of gauging feasibility for use in real-world production codes.

II. Tracking algorithm

We begin with a simplified abstract representation of the basic Monte Carlo transport algorithm. Let $P = \mathbb{R}^3 \times \mathbb{R}^+ \times \mathbb{S}^2 \times \mathbb{Z}$ denote the set of particles (neutrons) uniquely defined by a physical-space position $x \in \mathbb{R}^3$, energy $e \in \mathbb{R}^+$, direction $\Omega \in \mathbb{S}^2$ (where $\mathbb{S}^n$ denotes the n-sphere), and particle tag $z \in \mathbb{Z}^+$. Furthermore, let $B \subseteq P$ denote a countable subset of particles referred to as a *neutron batch*. A single *outer iteration* of the MC algorithm tracks each particle in a batch one-by-one through a sequence of collisions from birth to death (absorption or leakage from domain). Each particle interaction at some position x_p requires the lookup of probability distributions to randomly sample a number of different quantities, e.g. the type of interaction (e.g. elastic scattering, absorption, inelastic scattering), the scattering angle, scattering energy, and so forth. The precise details of the calculated quantities are not important for the present analysis. The key issue is that at each interaction point the local macroscopic cross sections need to be calculated by summing over the microscopic cross sections for each nuclide in the given material region. The microscopic cross sections are then read one-by-one from the lookup tables. With hundreds of nuclides present in the fuel region, this is an expensive operation that can take up the majority (up to 85%) of the total execution time of a typical execution of the code. Once the macroscopic cross sections are calculated each event can be sampled probabilistically.

The macroscopic cross section calculation process can be described more precisely as follows. Let $I = \{0, 1, ..., n-1\}$ denote the set of all n nuclides and $M = \{0, 1, ..., k-1\}$ denote the set of all k material *regions* in the reactor core (both are identified by an integer tag). Let the *atomic density function* $\rho : M \times I \to \mathbb{R}$ denote the atomic density of a given nuclide in a given material region, and let $G : \mathbb{R}^3 \to M$ denote the material lookup function— i.e. G selects the material region associated with a given particle position $x_p \in \mathbb{R}^3$. Note, $\rho(j, m) = 0$ for a nuclide j that is not present in a given material region m. Finally, define a *microscopic cross section table* for nuclide $j \in J$ as an element of $(\mathbb{R}^+)^{NE(j)}$, where $NE(j)$ denotes the number of tabulated cross section energy levels for nuclide j. If we similarly let $J = \{0, 1, 2, ..., l-1\}$ denote the set of relevant interaction types for a given simulation, then, Algorithm 1 shows a high level description of the key components of the conventional particle tracking algorithm.

The key operation in Algorithm 1 occurs at line number 6. It requires a separate load operation at each iteration of the inner loop— specifically, a table lookup of the microscopic cross section for nuclide j for interaction type i at energy level E. This is costly for several reasons. First, a typical neutron undergoes dozens of interactions and surface crossings 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 patterns yield little cache reuse and are unfriendly to data prefetching. Some nuclides require more than 100,000 tabulated energy levels for a single interaction type at a single temperature value. With a dozen reaction types, hundreds of nuclides and approximately fifty thermal energy levels required for an accurate calculation, the tabulated cross section data can exceed 100 GB of memory.[9] And though neutrons tend to travel from higher to lower energies (thermal upscattering is possible in LWRs), the precise cross section values jump around from interaction to interaction and are unpredictable in their details.

Algorithm 1 Classic Monte Carlo algorithm

```
1:  for $p \in B$ do                                  ▷ for each particle in batch
2:      repeat                                        ▷ until particle is absorbed
3:          $m \leftarrow G(x_p)$                     ▷ lookup material at $x_p$
4:          for $i \in I$ do                          ▷ for each nuclide
5:              for $j \in J$ do                      ▷ for each interaction type
6:                  $\sigma \leftarrow F(x_p, i, E_p)$        ▷ lookup microscopic xs
7:                  $\Sigma \leftarrow \Sigma + \rho(m, i)\sigma$   ▷ accumulate macro xs
8:              end for
9:              randomly sample interaction
10:             update particle properties
11:         end for
12:     until $absorbed(p)$
13: end for
```

Algorithm 2 Tracking processors in EBMS algorithm

```
1:  partition $[E_0, E_{max}]$ into $nb$ energy bands
2:  for $0 < n \leq nb$ do
3:      receive $\sigma(E_{n-1} : E_n)$ ▷ get x-section data from memory processor
4:      for $p \in B_n$ do                            ▷ for each particle in band $n$
5:          repeat  ▷ until particle is absorbed or $E(p) < E_{n-1}$
6:              $m \leftarrow G(x_p)$                 ▷ lookup local material
7:              for $i \in I$ do                      ▷ for each nuclide
8:                  for $j \in J$ do                  ▷ for each interaction type
9:                      $\sigma \leftarrow F(x_p, i, E_p)$ ▷ lookup microscopic xs
10:                     $\Sigma \leftarrow \Sigma + \rho(m, i)\sigma$ ▷ accumulate macro xs
11:                 end for
12:                 randomly sample interaction
13:                 update particle properties
14:             end for
15:         until $absorbed(p)$ or $energy(p) < E_n$
16:     end for
17:     $n \leftarrow n - 1$
18: end for
19: send termination signal to memory processors
```

III. The energy band memory server algorithm

Given the size of the neutron cross section tables, it is natural to seek a decomposition of cross section data across processors to reduce the on-node memory footprint. A naive implementation of this approach, however, is not likely to perform adequately in a production code. The particle history method is typically parallelized by carrying out Algorithm 1 independently on each processing core for a subset of the neutron batch B (and then synchronizing tallies at the end of each batch). When done this way, there is no affinity of the particles on a given processor to any subset of the cross section data, and thus this approach would require extremely frequent off-node communication of small amounts of data and and thus be performance limited by the off-node latency. Ideally, a decomposition strategy could be developed that both limited local cross section memory storage as well as minimized off-node read requests (i.e. that optimized the proximity of the data to the neutrons).

Our proposed approach, which we call the energy band memory server algorithm (EBMS), exploits 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 the thermal regime (.025 eV). In an LWR this path is for the most part unidirectional— i.e. particles move from higher to lower energies (some upscattering occurs in the thermal energy regime). The EBMS algorithm partitions the cross section data into *nb* energy *bands* $E_{nb}, E_{nb-1}, ...E_1$ and distributes them across the nodes of a distributed memory machine (different choices for decomposition strategy are discussed in the next section). The cross section data is no longer local to the node, and the new tracking algorithm requires moving the data in bands to the tracking processors. As depicted in Algorithm 2 above, each processor starts by loading the highest energy band and then tracking each particle one-by-one until it is either absorbed or leaves the energy band. In the latter case the tracking is resumed when the appropriate energy band is loaded. The process continues until all particles are absorbed, so that if upscattering occurs an outer iteration may be required to complete the simulation. In practice this will be uncommon and we expect 2-3 loop iterations maximum to fully complete the algorithm.

IV. Analysis of EBMS

In this section, we derive an expression for the expected total runtime of the EBMS algorithm. The goal of a Monte Carlo neutron problem is to track p neutrons through the material domain from generation until absorption or departure from the domain. This number p can be in the millions or billions for the highest fidelity simulations. Organizing the processing and memory resources for this task in the most efficient way is the ultimate goal of the application developer. The task requires directing a complex data flow of reading cross section data, recording interaction tallies, and updating neutron properties in memory.

Let M be the total size of cross section data measured in GB to be used in the application. We consider this parameter fixed for each problem and dependent on the particular physical regime and the computational accuracy required. One obvious advantage of the EBMS setup is its ability to enable more accurate neutronic simulations by accommodating cross section data sets larger than the on-node memory limit. Even in simplified applications where M is less than the on-node memory limit, there are advantages to not replicating the cross section data across processors; for example, avoiding replication frees memory for storing tallies and processing larger neutron batches.

We partition the cross section data into r equally sized *energy bands*, assign each band to a memory processor, and group memory processors into *memory clusters*. Assuming that all processors have the same memory limit, we note that the size of the bands in GB should not consume all of the memory of a processor, since the receiving tracking processors will need free memory for particles and tallies. Therefore, a more judicious use of memory processors would be to assign several energy bands to each processor in order to use up all the memory (each memory processor could own bands at opposite ends of the energy spectrum in order to avoid communication overlap). However, this would lead to more complicated contention in message passing. To avoid these considerations, we can assume that a cross section band takes up > 50% of on-node memory and each memory processor in a cluster owns one band.

Let n be the total number of MPI processes available to the application user. In some cases, each MPI process will correspond to a separate physical node, though we may also consider a process to be an individual processor core on a shared memory node. In the classic parallel Monte Carlo algorithm without cross section decomposition, the n processes independently track particles at some average rate $R = \left[\frac{time}{particle}\right]$. Excluding essential inter-process communication (including reductions and synchronizations), the overall runtime is approximated by

$$T_{classic} \approx \frac{Rp}{n}$$

where Rp is the serial tracking time for p particles.

For a fair evaluation of the EBMS algorithm, we consider the number of MPI processes n fixed; every added memory processor comes at the cost of a tracking processor. Let m be the number of memory clusters, each consisting of r processors, for a total number of memory processors mr. The total number of tracking processors is then $n - mr$ and the runtime of the algorithm can be described by

$$T_{EBMS}(m,r) \approx \frac{Rp}{n - mr} + \max_{i \le n-mr} K_i\left(\frac{n - mr}{m}, M, r, \alpha, \beta\right)$$

where K_i is a function representing the total cost of cross section data movement to the ith tracking process. The first term simply represents the penalty associated with reallocating tracking processors as memory processors and is strictly increasing while $\max(K_i)$ is strictly decreasing in r and m.

Predicting the precise cost of data movement is complex. In each memory cluster each associated tracking processor needs to move at minimum one band of data from each memory server (this can be more depending on how many outer iterations are required to convergence). The intrinsic cost of each data movement can be modeled approximately as $\alpha + \beta M/r$, where α and β are appropriate values of application-level latency and inverse bandwidth, respectively. Since there are r bands, the total cost of data movement for each processor is then $\alpha r + \beta M$.

However, this must be seen as a lower bound and certainly will never be attained in practice— in reality multiple tracking processors will be attempting to obtain the the same memory band simultaneously, and application-level contention will significantly erode performance. In fact, it is easy to derive the

symbol	description
α	inter-node point-to-point latency
β^{-1}	inter-node point-to-point bandwidth
r	energy bands per cluster
m	number of memory clusters
n	total number of processors
p	total number of particles
M	total cross section memory
R	serial tracking rate in $[\frac{time}{particle}]$

Table 1: Summary of variables used in analysis

worst case scenario for this condition by assuming that access to each memory process is serialized. That is, each tracking processor in a cluster will simultaneously attempt to access the same memory server, and each tracking processor will have to be served in turn. Given that there are $\frac{n-mr}{m}$ tracking processors associated with each memory server cluster, the worst contention would be written as

$$K_{worst} = (\alpha r + \beta M)(\frac{n-mr}{m})$$

Thus, the EBMS runtime is

$$T_{EBMS}(m,r) = \frac{Rp}{n-mr} + \zeta(\alpha r + \beta M) \qquad (1)$$
$$1 \leq \zeta \leq \frac{n-mr}{m}$$

Equation 1 bounds the performance of the algorithm and predicts the optimum number of clusters for a given problem size and architecture. By minimizing the upper bound over the number of memory clusters, the application user can ensure a reasonable performance cost; however, the upper bound minimum is unlikely to coincide with the actual runtime minimum given the complex nature of network and application contention. Table 1 summarizes the relevant parameters used in the performance model.

A few observations are readily apparent in the form of equation 1. We first note that the latency term, $\zeta\alpha r$ is negligible relative to all other terms in practical situations. For a typical network latency on the order of 10^{-6} seconds, the simulation would need to employ hundreds of thousands of processors and energy bands to generate an appreciable performance cost. Even in such a large simulation, the latency cost would likely be dominated by the tracking time. Second, we observe that the optimal number of memory clusters m is typically located far away from the extremes of the interval $[1, n/mr]$ and largely depends on the ratio of $Rp/(\beta M)$.

To evaluate the model, we consider parameters in the space of interest of large-scale reactor analysis. In this example, 1000 nodes of a Blue Gene/Q system were considered for a 250 million particle, 100 GB cross section computation. The Blue Gene/Q installation has an effective unidirectional point-to-point bandwidth β^{-1} = 1.8 GB/s and latency α = 3.53 μs.**[10]** Figure 1 plots the expression $\frac{T_{EBMS}}{T_{classic}}$ for the upper and lower bounds of the performance model over a range of memory cluster counts. The model predicts reasonable performance of the EBMS algorithm. For both 10 and 30 energy bands (10, 3.33 GB per memory process, respectively), the optimal upper bound is less than 5 times slower than the classic algorithm.

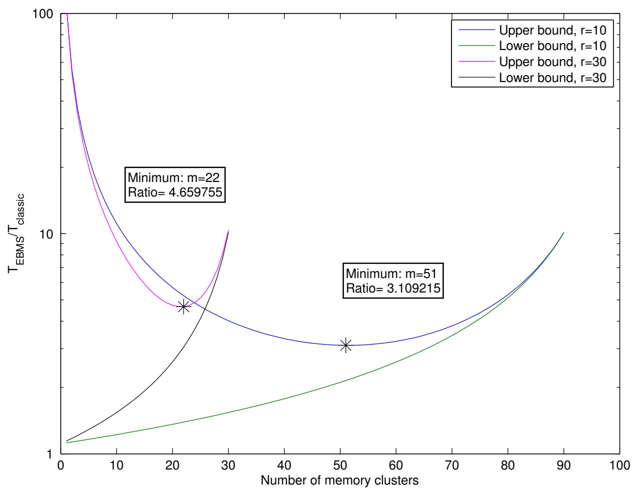

Figure 1: Performance model upper and lower bounds for a large neutronics problem on the Blue Gene/Q architecture (n=1000, r=10, M=100 GB, R^{-1} = 500 particles/sec, p=250 million)

Although the gap between the upper and lower bound is high and thus the model is potentially impractical, one interesting feature is that the gap is not large near the minimum. Therefore, the model is a useful metric for bounding the optimal performance. Further, the upper bound approaches the lower bound at large memory cluster counts. The model is valid for $\frac{n-mr}{m} \geq 1$ (1 tracking process per cluster), so the communication term in the upper bound approaches the constant $(\alpha r + \beta M)$.

V. Numerical experiments

We seek to evaluate the accuracy of our performance model's upper and lower bounds over a range of reasonable application parameters. Additionally, we want to optimize the number of energy bands and memory clusters to minimize the performance cost when compared to the classic replication strategy. Finally, we evaluate the reasonability of such a cost when weighed against the memory savings of EBMS.

1. Proxy application

To carry out our experiments we develop a simplified MC proxy application that mimics the workload of the full algorithm. It uses homogenized data from the Monte Carlo solver, OpenMC,**[11]** to represent the key features of the full tracking algorithm. The application is written in C/MPI and takes as input the number of particles, the size of the cross section data array, the number of energy bands, and the number of memory clusters.

The application divides the tracking processors equally among the memory clusters. Each tracking processor at the start of each energy band posts an `MPI_Recv()` for the corresponding memory processor in its cluster and waits for the request for cross section data to be fulfilled. The tracking processors then produce tracking rates by integrating each particle through a series of randomly sampled interactions in a homogeneous material. At each interaction particle absorption is determined

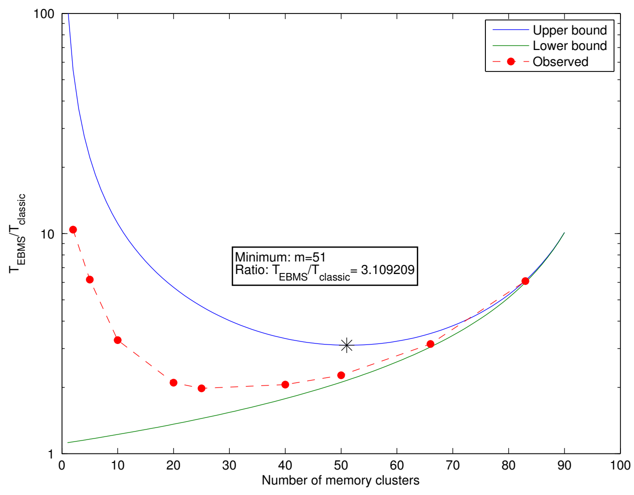

Figure 2: Proxy application performance results from Vesta compared to the performance model for R^{-1} = 500 particles/sec

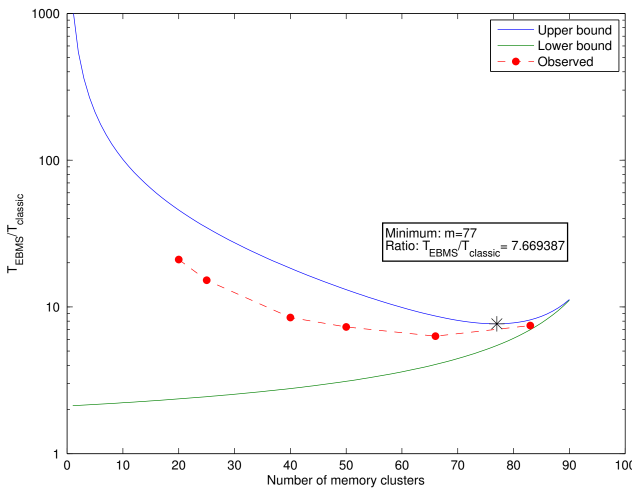

Figure 3: Proxy application performance results from Vesta compared to the performance model for R^{-1} = 5000 particles/sec

probabilistically by a gross absorption rate probability, and energy group scattering probabilities are determined by an $r \times r$ user-input scattering matrix, which is derived from execution of OpenMC on a classic reactor benchmark calculation.(12)

2. Results

The proxy application was executed on Argonne National Laboratory's Blue Gene/Q "Vesta" system. We examined the 10 energy band case of Figure 1 with $n = 1000, r = 10, M = 100$ GB, $p = 250$ million neutrons. In the interest of avoiding complex intra-node memory considerations, the tests ran with 1 MPI task/node. Figures 2, 3 plot the number of memory clusters versus performance as the ratio over the classic algorithm.

In seeking to validate the model, we find that the empirical results are in strong agreement with the predictions derived in the previous section. The optimal performance of the proxy application in Figure 2 was roughly twice that of the classic algorithm, thus significantly excelling the worst-case prediction. In general, the upper bound is seen to be highly conservative on the Blue Gene/Q architecture. The gap between the upper bound and actual performance grows as the problem size becomes smaller due to the increased weight of the communication term relative to the tracking term. Nevertheless, the model establishes a reliable upper bound for the total runtime that is independent of the underlying network layout and only depends on the maximum point-to-point machine bandwidth and latency.

The tracking rate of 500 particles/sec is a conservative figure derived from the observed performance of the OpenMC code in LWR analysis.(7) In high fidelity or multiphysics simulations, one can expect slower tracking rates and thus smaller relative communication costs in EBMS. However, it is worth considering how flexible the algorithm is for faster tracking rates. Figure 3 shows the model predictions for the same problem but an with an order of magnitude faster tracking rate. The model becomes more pessimistic as the local tracking work becomes less significant. However, we again observe that the proxy application performance fares far better than the worst-case prediction. Therefore, while it is always in the application user's interest to maintain a large ratio of local work Rp to data movement work βM, the actual performance cost of EBMS remains reasonable over a wide range of tracking rates typical in LWR analysis.

VI. Conclusion

We have demonstrated a flexible algorithm which exploits basic properties of neutron physics to make more efficient reuse of memory in neutral particle Monte Carlo transport simulations. The algorithm can adjust to a range of cross section memory footprint sizes so that the user can optimally tune it for their particular application. 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 code.

An analytic performance model based on the simulation parameters and network characteristics was created for the algorithm. Our results indicated that the bounds for the best and worst-case performance are well-established and serve as useful metrics near the performance minimum.

We note that there exist several potentially simple algorithmic improvements. For example, creating hybrid tracking/memory processors and allowing tracking processors to access any available memory cluster are application-specific strategies that might yield significant speedup. Nevertheless, this study shows that the most basic implementation of the energy band memory server algorithm is a worthwhile approach to managing the memory burden while maintaining reasonable performance. Future work with EBMS will examine the complex concerns of topology-aware mappings and network contention in exascale architectures.

Acknowledgment

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

References

1) 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.
2) K. Felker, A. Siegel, S. Siegel, Optimizing memory constrained environments in monte carlo nuclear reactor simulations, International Journal of High Performance Computing Applications 27 (2) (2013) 210–216. doi:10.1177/1094342012445627.
3) F. Brown, Recent advances and future prospects for monte carlo, Tech. Rep. LA-UR-10-05634, Los Alamos National Laboratory (2010).
4) D. C. Irving, R. M. F. Jr., F. Kam, O5R, A general-purpose Monte Carlo neutron transport code, Tech. rep., Oak Ridge National Laboratory (1965).
5) 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.
6) 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.
7) P. K. Romano, A. R. Siegel, B. Forget, K. Smith, Data decomposition of Monte Carlo particle transport simulations via tally servers, J. Comput. Phys. 252 (2013) 20–36.
8) Q. Niu, J. Dinan, S. Tirukkovalur, L. M. tas, L. Wagner, P. Sadayappan, A Global Address Space Approach to Automated Data Management for Parallel Quantum Monte Carlo Applications, in: Proc. 19th Intl. Conf. on High Performance Computing (HiPC), Pune, India, 2012.
9) 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.
10) D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, A. Choudhury, Y. Sabharwal, S. Singhal, B. Steinmacher-Burow, J. Parker, Looking under the hood of the BM Blue Gene/Q network, in: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, 2012, pp. 1–12. doi:10.1109/SC.2012.72.
11) 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.
12) 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