Nuclear Engineering and Technology 49 (2017) 1165–1171

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

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

Original Article

Limits on the efficiency of event-based algorithms for Monte Carlo neutron transport

Paul K. Romano*, Andrew R. Siegel*

Argonne National Laboratory, Mathematics and Computer Science Division, 9700 S. Cass Avenue, Lemont, IL 60439, USA

ARTICLE INFO

Article history:
Received 31 May 2017
Accepted 22 June 2017
Available online 1 July 2017

Keywords:
Monte Carlo
Neutron transport
Parallelism
Vectorization

ABSTRACT

The traditional form of parallelism in Monte Carlo particle transport simulations, wherein each individual particle history is considered a unit of work, does not lend itself well to data-level parallelism. Event-based algorithms, which were originally used for simulations on vector processors, may offer a path toward better utilizing data-level parallelism in modern computer architectures. In this study, a simple model is developed for estimating the efficiency of the event-based particle transport algorithm under two sets of assumptions. Data collected from simulations of four reactor problems using OpenMC was then used in conjunction with the models to calculate the speedup due to vectorization as a function of the size of the particle bank and the vector width. When each event type is assumed to have constant execution time, the achievable speedup is directly related to the particle bank size. We observed that the bank size generally needs to be at least 20 times greater than vector size to achieve vector efficiency greater than 90%. When the execution times for events are allowed to vary, the vector speedup is also limited by differences in the execution time for events being carried out in a single event-iteration.

1. Introduction

At present, most commodity CPUs feature instruction set architectures that can utilize vector instructions on multiple data sets coupled with a vector processing unit (VPU) that can process instructions in a single clock cycle. This is especially true of CPUs tailored for numerical computations, such as Intel's Xeon Phi processor, which features a 512-bit wide VPU. Consequently, programmers must effectively expose data-level parallelism in their applications in order to maximize the performance on these architectures. While many scientific codes can be optimized for such architectures with modest changes (e.g. modifying data layout and alignment or reordering loops), data-level parallelism poses a problem for Monte Carlo (MC) neutron transport because it is characterized by the frequent use of conditional branching, random memory access patterns, and performance that is often limited by memory latency due to high cache miss rates.

Researchers agree that significant changes in algorithms and data structures will be necessary in order to exploit vectorization in MC neutron transport simulations. One approach that has been attempted is to restructure the transport algorithm from one that is *history based*, wherein the basic unit of work is the history of a single particle from birth to death, to one that is *event based*, wherein the unit of work is a particular *event* within a single particle's history. What constitutes as an event may differ from one implementation to another, but generally events can be categorized into actions such as a collision with a target nuclide, the free flight of a particle between two spatial locations, or a particle crossing a material interface. Event-based algorithms were initially proposed and studied [1,2] in the 1980s when the largest supercomputers at the time relied on vector processors. The algorithms were implemented in several codes, including a general geometry, continuous-energy MC code [3].

The past few years have seen a resurgence of interest in event-based algorithms inspired by the higher levels of data-level parallelism inherent in two products aimed at high-performance computing—the Intel Xeon Phi processor and the NVIDIA Tesla GPU. Three recent studies considered vectorization of one-dimensional, one-group MC neutron transport codes [4–6]. Early attempts have also been made at employing event-based algorithms for more general, continuous-energy MC codes targeting GPU architectures [7,8]. In addition to the efforts in the nuclear engineering community, we note that similar algorithms are being explored in high-energy physics [9–11].

* Corresponding authors.
E-mail addresses: promano@anl.gov (P.K. Romano), siegela@mcs.anl.gov (A.R. Siegel).

http://dx.doi.org/10.1016/j.net.2017.06.006

As discussed in [1], restructuring a Monte Carlo code to employ an event-based algorithm is not a trivial change because all the data structures and loop organization need to be modified. Often, architecture-specific optimization is also necessary in order to attain optimal performance. The goal of the present study is not to look at a specific implementation of event-based algorithms but rather to study them from a theoretical standpoint. Specifically, this study seeks to determine practical limits on the efficiency of event-based algorithms based on data flow within a hypothetical implementation. Coupled with data from a full-scale Monte Carlo code, OpenMC [12], we can quantify limits without the need to implement an event-based algorithm in a real code. We view this as an easier first step toward understanding the performance of the event-based algorithm if it were to be implemented in a vectorized general geometry, continuous-energy Monte Carlo code.

2. Theory

Suppose we are simulating N neutrons. Each neutron history can be viewed as a sequence (or queue) of events. Let us denote the jth event of the ith particle by $\chi_{i,j}$. Thus, we represent the complete history of the ith neutron as a queue of events that are to be processed,

$$X_i = \left(\chi_{i,1}, \chi_{i,2}, \ldots, \chi_{i,M}\right), \tag{1}$$

where M is the total number of events. We can also view the entire simulation itself as a queue of particles (event queues) to be simulated. Let us denote all particles by

$$\mathbf{X} = (X_1, X_2, \ldots, X_N). \tag{2}$$

In the history-based algorithm, the event queue for each particle is processed in order and parallelization can be achieved over particles. In the event-based algorithm, a single parallel task keeps track of multiple particles simultaneously. N is generally assumed to be sufficiently large that storing the attributes of every particle simultaneously is not feasible. Consequently, the number of particles "in flight" at a given time is limited to at most B particles. We refer to a queue of B particles that are to be simulated by a single parallel task as a *particle bank*. This is directly analogous to the "initial particle vector" in [2].

As discussed in [1] and [2], there are two main variations on how particle data is collected in vectors to be processed: the *basic* approach and the *stack-driven* approach. In the basic approach, the program flow follows the history-based algorithm wherein the events for each particle are processed sequentially. At any given point during the simulation, the particles within a bank are executing the same event, although some may be masked because they either do not participate or have already been killed off. In the stack-driven approach, a stack is created for each event type, and particles are pushed onto or popped off of the stacks as the event in each particle queue changes. Let $T(\chi)$ be the type of event χ and $\tau(\chi)$ be the execution time for event χ. Let $\mathbf{\Gamma}$ be the set of all particle stacks. We denote the length of a queue or stack by $|\cdot|$.

The stack-driven event-based algorithm is prototyped in Algorithm 1. The algorithm begins by distributing B particle event queues to the three stacks based on the type of the next event to be processed for each particle. Following this is an iteration where the events for particles in the longest stack are processed. As each event is processed, particles are redistributed to the stacks based on the next event type. The event iteration continues until each of the three stacks is empty. At this point, new event queues are distributed to the three stacks, and the event iteration starts over. The whole process repeats until no event queues remain. Note that the Pop/Push and Enqueue/Dequeue operations on stacks and queues, respectively, follow textbook definitions.

Algorithm 1 Stack-driven event-based algorithm

```
while |X| > 0 do                                   ▷ While particles remain
    Distribute(X, Γ, B)
    while |Γ_E| > 0 for at least one event E do
        Determine longest particle stack Γ_E
        ProcessEvents(Γ, E)
    end while
end while

function Distribute(X, Γ, B)
    for i ← 1 to B do
        X_k ← Dequeue(X)
        E ← T(χ_{k,1})                             ▷ Get type of first event
        Push(Γ_E, X_k)                             ▷ Push particle to appropriate stack
    end for
end function

function Process(Γ, E)
    for i ← 1 to |Γ_E| do
        X_k ← Pop(Γ_E)
        χ ← Dequeue(X_k)
        Process event χ
        if |X_k| > 0 then
            E' ← T(χ_{k,1})                        ▷ Determine type of next event
            Push(Γ_{E'}, X_k)
        end if
    end for
end function
```

The objective of our analysis is to estimate the time to solution for the stack-driven event-based algorithm relative to the history-based algorithm. In the event-based algorithm, the time to solution may be reduced because events from multiple queues can be processed simultaneously. Let us assume that the computer architecture we are simulating on has a VPU capable of performing V floating-point operations simultaneously. In the history-based method, there are no vectorization gains, and thus the time to solution is

$$t_H = \sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \sum_{h=1}^{|\Gamma_{k,\ell}|} \tau\left(\chi_{k,\ell,h}\right), \tag{3}$$

where k corresponds to the outer loop in Algorithm 1, ℓ corresponds to the inner loop in Algorithm 1, L_k is the total number of inner iterations for outer iteration k, $\Gamma_{k,\ell}$ is the stack for a given outer/inner iteration, and $\chi_{k,\ell,h}$ is the event at the front of the hth event queue. Eq. (3) simply states that the total execution time when using the history-based method is the sum of the execution times for all events. To estimate the execution time for the event-based algorithm, we will consider two sets of assumptions:

- Assume that all events of a given type execute in exactly the same amount of time. We will refer to this as **case 1.** If the stack length at a given stage is an exact multiple of V, then that stage can be completed V times faster than when using the history-based method. At any given stage, however, the stack length may not be an exact multiple; thus, there will be a so-called remainder loop wherein some number of events less than V are processed simultaneously.

- Assume that V events can be processed simultaneously but that the execution time is limited by the time to execute the longest event within those V events. We will refer to this as **case 2.** In this case, the speedup from vectorization is limited both by remainder loops and the distribution of event execution times.

In the first case, we can express the time to solution of the event-based algorithm as

$$t_1 = \sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \left\lceil \frac{|\Gamma_{k,\ell}|}{V} \right\rceil \tau_E, \tag{4}$$

where E is the event type for the ℓth event iteration in outer iteration k and τ_E is the average time to process event type E. In the second case, the expression becomes slightly more complicated:

$$t_2 = \sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \sum_{g=1}^{\lceil |\Gamma_{k,\ell}|/V \rceil} \max\left(\tau\left(\chi_{k,\ell,h}\right)\right), \tag{5}$$

where the maximum is taken over values of $\tau(\chi_{k,\ell,h})$ such that $h \in [(g-1)V + 1, \min(gV, |\Gamma_{k,\ell}|)]$. For example, if $V = 4$ and $|\Gamma_{k,\ell}| = 8$, the total time to process the 8 events within the stack would be the sum of the maximum execution time for the first four events and the maximum execution time for the last four events. Under the first set of assumptions, the effective speedup due to vectorization is

$$\eta_1 \equiv \frac{t_H}{t_1} = \frac{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \sum_{h=1}^{|\Gamma_{k,\ell}|} \tau\left(\chi_{k,\ell,h}\right)}{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \left\lceil \frac{|\Gamma_{k,\ell}|}{V} \right\rceil \tau_E} = \frac{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} |\Gamma_{k,\ell}| \tau_E}{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \left\lceil \frac{|\Gamma_{k,\ell}|}{V} \right\rceil \tau_E}. \tag{6}$$

If $\Gamma_{k,\ell} \bmod V = 0$ for all k and ℓ, then $\eta_1 = V$, which represents the maximum speedup possible. Under the second set of assumptions, the effective speedup is

$$\eta_2 \equiv \frac{t_H}{t_2} = \frac{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \sum_{h=1}^{|\Gamma_{k,\ell}|} \tau\left(\chi_{k,\ell,h}\right)}{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k} \sum_{g=1}^{\lceil |\Gamma_{k,\ell}|/V \rceil} \max\left(\tau\left(\chi_{k,\ell,h}\right)\right)}. \tag{7}$$

Because η_2 accounts for both remainder loops and differences in event times whereas η_1 accounts only for remainder loops, we have that $\eta_2 \leq \eta_1 \leq V$.

Note that we have made a few crude approximations in the development of Eqs. (6) and (7). We assumed that the differences in event times directly relate to the ability to vectorize the events. In reality, the ability to vectorize an event may not be well characterized by differences in event times for a serial (history-based) code. Nevertheless, evaluating the efficiency in this manner can give us an idea of how the efficiency might change if a particular event is not perfectly vectorizable. We have also assumed that there is no cost to data movement in the event-based algorithm and that there is no latency for performing vector operations. While these assumptions do not hold for a realistic simulation, they allow us to establish an upper bound on the efficiency of the algorithm since removing the assumptions would serve only to reduce the efficiency.

3. Results and analysis

To evaluate the two estimates for the vector speedup of the event-based algorithm, η_1 and η_2, we used a modified version of OpenMC to collect event queues, **X**. For each event within a queue, we store the type of the event and the execution time of that event during the simulation. Following the work of Martin and Brown [2], we consider three types of events: free flight (F), collision (C), and boundary crossing (B). Of course, the event times and list of events will strongly depend on the model being simulated. Thus, we consider four reactor models for this study:

1. The BEAVRS light-water reactor (LWR) benchmark [13], which represents an application (LWR simulation) with formidable computational requirements that could benefit from high-performance computing architectures.
2. The BEAVRS model with the fuel composition modified to include about 250 nuclides. This model has computational requirements more representative of a depletion simulation.
3. A sodium fast reactor (SFR) benchmark problem based on the prototype Generation-IV SFR concept [14,15]. An SFR model was chosen so that both fast and thermal spectra were covered.
4. A very high temperature reactor (VHTR) prismatic assembly based on a benchmark problem developed at Argonne National Laboratory [15]. A VHTR model was chosen because the inclusion of a graphite moderator changes the neutron physics considerably; that is, the average neutron suffers many more collisions in a graphite-moderated system than for other moderators.

Figs. 1–3 show the geometry of the BEAVRS, SFR, and VHTR models, respectively.

The event and timing data that results from each simulation was then read by a Python script that implements Algorithm 1 and keeps track of the estimated time to solution of the event-based and history-based algorithms via Eqs. (3)–(5). At each event iteration, it calculates the relative time of the event-based and history-based algorithms as

$$\eta_{1,\ell} = \frac{t_{H,\ell} - t_{H,\ell-1}}{t_{1,\ell} - t_{1,\ell-1}} \tag{8}$$

$$\eta_{2,\ell} = \frac{t_{H,\ell} - t_{H,\ell-1}}{t_{2,\ell} - t_{2,\ell-1}}, \tag{9}$$

where $t_{H,\ell}$, $t_{1,\ell}$, and $t_{2,\ell}$ are the accumulated values of t_H, t_1, and t_2, respectively, at the ℓth event iteration. The script allows any

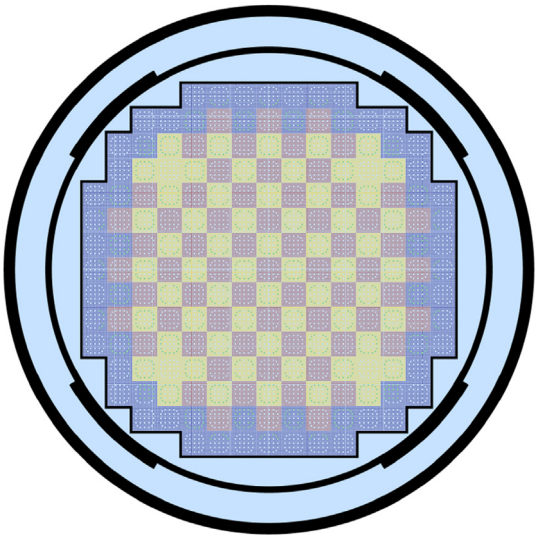

Fig. 1. BEAVRS model geometry.

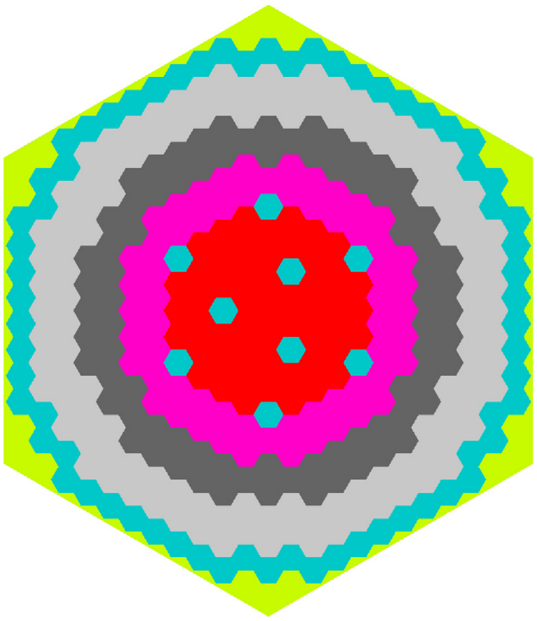

Fig. 2. Sodium fast reactor model geometry.

arbitrary choice of V and B. All the results for this study are for $V \in \{2, 4, 8, 16\}$ and $B \in \{16, 32, 64, 128\}$.

3.1. BEAVRS results

To develop an intuition for how the performance might behave with different choices of the parameters, we present in Fig. 4 the average length of the longest stack relative to the vector size at each event iteration in Algorithm 1 for $V = 8$ and each choice of B. We see that regardless of the particle bank size, the average number of events to finish simulating all B particles is about the same. This reflects the fact that the average number of events is directly related to the physical properties of the system rather than the arrangement in which we process them. The vector efficiency of any given iteration is effectively $S/\lceil S \rceil$, where $S = |\Gamma_\ell|/V$. Thus, the best situation occurs if $|\Gamma_\ell|$ is exactly a multiple of V. The worst efficiency generally occurs where $S < 1$. We see in Fig. 4 that the longest stack length begins to dip below V after 400 iterations when $B = 16$. When the bank size is increased to $B = 128$, it takes nearly

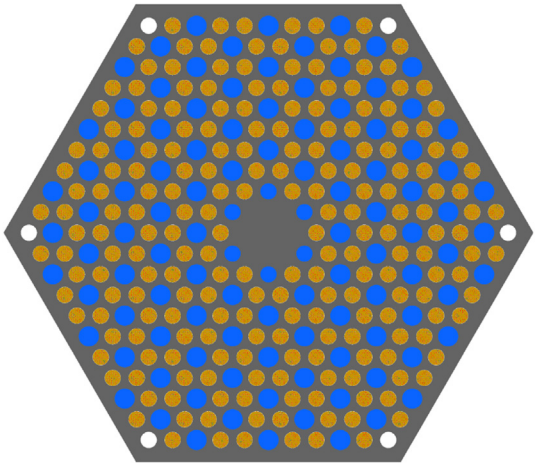

Fig. 3. Very high temperature reactor prismatic assembly geometry.

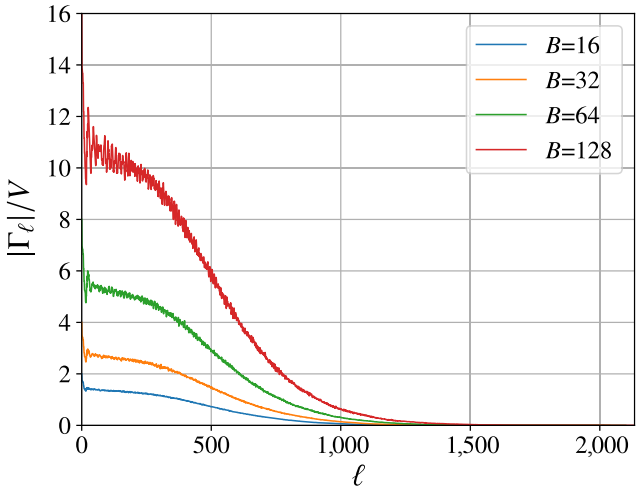

Fig. 4. Average stack size as a function of event iteration for $V = 8$ using the BEAVRS model.

1,000 event iterations to reach the same condition. Clearly, then, increasing the bank size should increase the overall vector efficiency because there are fewer iterations with $S < 1$.

When all events of a given type are assumed to execute in exactly the same amount of time (case 1), the overall vector efficiency would be solely a function of the stack length shown in Fig. 4. However, when we explicitly account for differences in event times for the same event type (case 2), the vector efficiency depends not only on the stack length but also on the differences in event times, as captured by Eq. (5). Fig. 5 shows the average speedup at a given event iteration for the two cases, $\eta_{1,\ell}$ and $\eta_{2,\ell}$, gain for $V = 8$. We see that for case 1, increasing the bank size brings the initial speedup close to the maximum value possible (V) and allows it to stay high for more event iterations. In case 2, however, no matter how large the bank size is, the average speedup at each event iteration is limited by the differences in event times. One can infer then that for this particular model an infinitely large particle bank would result in a vector speedup of no more than about 4 under the assumptions of case 2.

The final results for the vector speedup as a function of V and B for the BEAVRS model are shown in Fig. 6. For case 1 (solid lines),

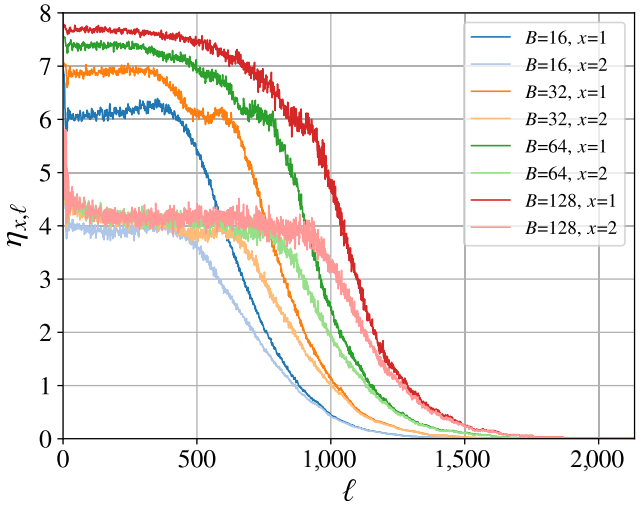

Fig. 5. Average speedup over history-based method as a function of event iteration for $V = 8$ using the BEAVRS model.

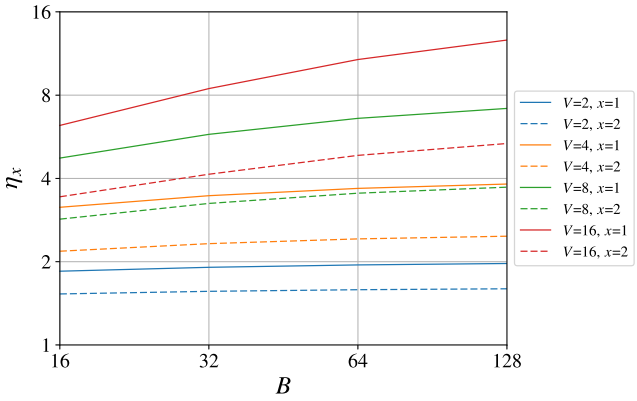

Fig. 6. Effective speedup of event-based algorithm as a function of bank size and vector size using the BEAVRS model.

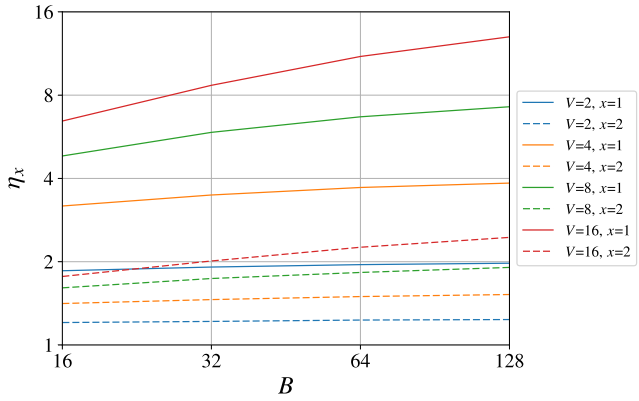

Fig. 8. Effective speedup of event-based algorithm as a function of bank size and vector size using the BEAVRS model with large fuel nuclide inventory.

one can see that as B increases, the vector speedup approaches the vector length. For case 2, however, the vector speedup asymptotically approaches some value less than V. This makes sense in light of the speedup at each individual event iteration shown in Fig. 5. We also see that as V increases, the difference between the maximum speedup and the asymptotic speedup for case 2 grows larger.

Fig. 6 shows that for $V = 2$, the vector speedup for case 1 is almost exactly 2 for a bank of 128 particles. As the vector size is increased, the bank size also needs to be larger in order to obtain good efficiency. This can be seen more clearly in Fig. 7, which displays the vector efficiency η/V as a function of the bank-to-stack size ratio. For both case 1 and case 2, the initial bank size needs to be at least 30 times the vector size in order to reach the asymptotic speedup.

Fig. 8 shows the calculated vector speedup for the BEAVRS model with large fuel nuclide inventory. The primary difference in this model is that free-flight events, which include the lookup of cross sections, have the longest average execution time of the three events. Furthermore, the distribution of free-flight event execution times has a large variance due to considerable differences in the number of nuclides in different materials. The overall effect is that while the vector speedup for case 1 is nearly the same as that observed for the original BEAVRS model, the speedup is much lower for case 2 where differences in event execution times are accounted for.

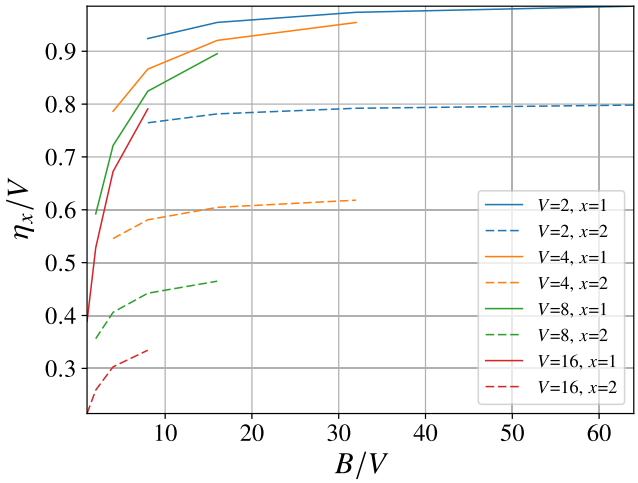

Fig. 7. Overall vector efficiency as a function of the bank to vector size ratio using the BEAVRS model.

3.2. SFR results

We turn our attention now to the SFR model. The primary difference between the SFR and the BEAVRS model is that the neutron spectrum is fast rather than thermal. Consequently, the mean free path of neutrons is longer, resulting in more boundary crossing and free flight events as opposed to collisions. Nevertheless, this doesn't appear to have much of an impact on the achievable vector speedup based on our models. Fig. 9 shows the vector speedup as a function of V and B for this model; we see that it is almost identical to the results in Fig. 6.

3.3. VHTR results

The last model analyzed was the VHTR prismatic assembly. The presence of a graphite moderator results in the neutrons having many more collisions on average when compared to an LWR or an SFR. In the context of the event-based algorithm, this means that a higher proportion of time is spent simulating collision events. In all reactor types, collisions tend to have high variability in their execution time because many different representations exist for secondary angle and energy distributions in the continuous-energy nuclear data. We therefore observe that the average speedup obtained for a single event iteration for case 2, shown in Fig. 10 for $V = 8$, tends to be lower for the VHTR model than for the BEAVRS or SFR models. The overall vector speedup for the VHTR as a function of V and B is shown in Fig. 11. Not only is the overall speedup low for case 2, but the speedup does not change as much as a function of B.

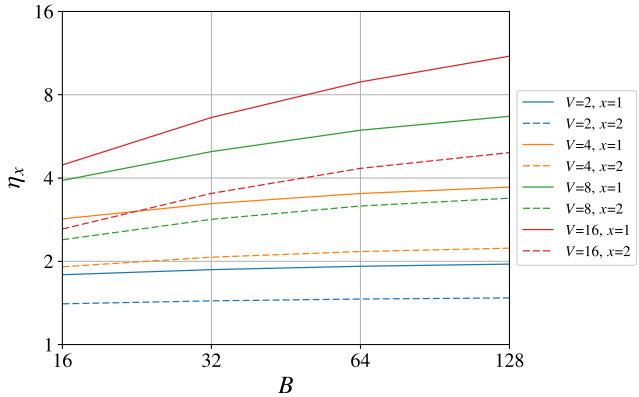

Fig. 9. Effective speedup of event-based algorithm as a function of bank size and vector size using the SFR model.

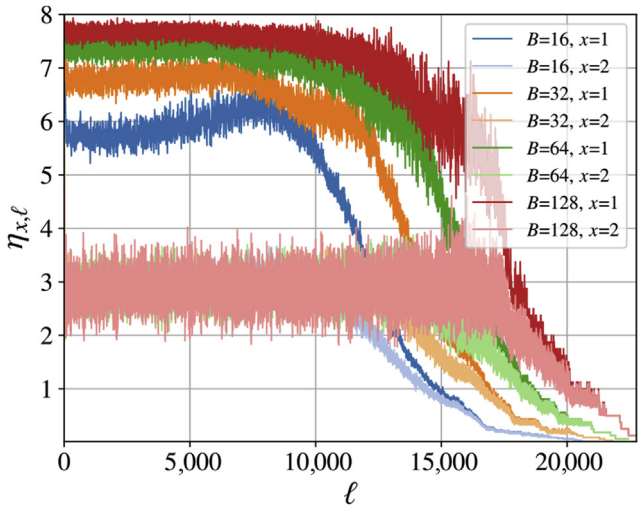

Fig. 10. Average speedup over history-based method as a function of event iteration for V=8 using the VHTR model.

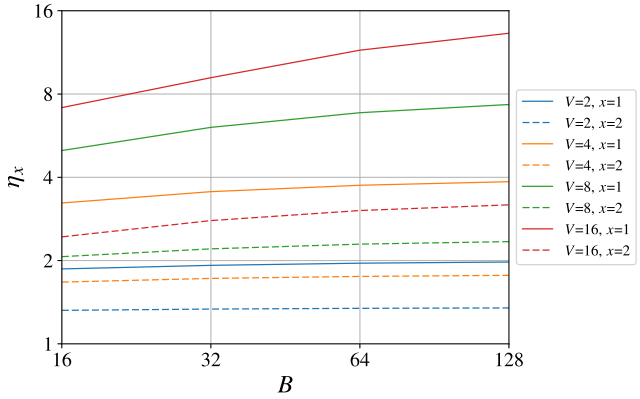

Fig. 11. Effective speedup of event-based algorithm as a function of bank size and vector size using the VHTR model.

The reason is that the high average number of collisions per particle ensures that the stack length remains high for many event iterations, as shown in Fig. 10.

4. Conclusions

We have developed a simple model for estimating the speedup from vectorization in the event-based particle transport algorithm under two sets of assumptions. First, we assumed that each event type has constant execution time. We then relaxed that assumption and allowed each event type to have a distribution of event times. These assumptions result in two different models for the vector speedup in Eqs. (4) and (5). Data collected from simulations of four reactor problems using OpenMC was then used in conjunction with the models to calculate the vector speedup as a function of two parameters: the size of the particle bank and the vector width. The results demonstrate that when events are assumed to have constant execution time, increasing the bank size is sufficient to obtain a speedup close to the maximum possible. We observed that the bank size generally needs to be at least 20 times greater than vector size in order to achieve vector efficiency greater than 90%. When the execution times for events are allowed to vary, however, the vector speedup is also limited by the differences in execution time for events being carried out in a single event iteration. For some problems (for example, the VHTR and the BEAVRS model with many nuclides in fuel), this implies that vector efficiencies over 50% may not be attainable.

We emphasize that this study is only a first step toward understanding the performance of event-based algorithms. A number of assumptions were made to arrive at simple expressions for the time to solution and would not hold for a realistic simulation. For example, we have assumed that there is no cost to data movement inherent in the event-based algorithm through the use of particle stacks. This aspect may be difficult to capture in a theoretical model. We have also completely neglected tallies. For large-scale simulations, tallying physical quantities can consume a significant fraction of the execution time, so any successful model or analysis must account for this. This aspect is sufficiently complicated that a focused study on tallies would be worthwhile.

Another limitation of our study is that we have looked at only one choice for how to divide particle histories into events; namely, we considered only three events: free flight, collision, and boundary crossing. In reality, each of these events is sufficiently complicated that it may make sense to use finer-grained events. When using coarse-grained events, not all the logic will be vectorizable, especially for collision events in a continuous-energy code because of the inherently deep branching logic. Our assumption that the execution time of an event iteration is limited by the maximum serial time of an individual event is perhaps not realistic. Notwithstanding, it does provide a means for modeling the fact that the entire event is not vectorizable. Our analysis points to this being a limiting factor in the achievable speedup.

One must keep in mind that data-level parallelism is only one aspect of achieving performance on modern computer architectures. While achieving good use of vectorization is often important in maximizing performance, other optimizations may be more crucial, especially for codes that tend to be limited by memory latency and bandwidth. MC neutron transport is one such application that has been characterized as being limited by the memory subsystem [16,17]. In practice, the use of event-based algorithms could yield better performance than the history-based algorithm by virtue of better cache utilization. Capturing such an effect in a theoretical model is notoriously difficult, however.

Acknowledgments

This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

References

[1] F.B. Brown, W.R. Martin, Monte Carlo methods for radiation transport analysis on vector computers, Prog. Nucl. Energy 14 (3) (1984) 269–299.
[2] W.R. Martin, F.B. Brown, Status of vectorized Monte Carlo for particle transport analysis, Int. J. Supercomput. Appl. 1 (2) (1987) 11–32.
[3] F.B. Brown, Vectorization of three-dimensional general-geometry Monte Carlo, Trans. Am. Nucl. Soc. 53 (1986) 283–285.
[4] X. Du, T. Liu, W. Ji, X.G. Xu, F.B. Brown, Evaluation of vectorized Monte Carlo algorithms on GPUs for a neutron eigenvalue problem, in: M&C, Sun Valley, Idaho, May 5–9 2013.
[5] T. Liu, X. Du, W. Ji, X.G. Xu, F.B. Brown, A comparative study of history-based versus vectorized Monte Carlo methods in the GPU/CUDA environment for a simple neutron eigenvalue problem, in: SNA + MC, Paris, France, 2014.
[6] P.S. Brantley, S.A. Dawson, M.S. Mckinley, M.J. O'Brien, D.E. Stevens, B.R. Beck, I.Eugene D. Brooks, Advanced computing architecture challenges for the

Mercury Monte Carlo particle transport project, in: Joint Int. Conf. on Mathematics and Computation, Supercomputing in Nuclear Applications, and the Monte Carlo Method, Nashville, Tennessee, Apr. 19–23 2015.
[7] R.M. Bergmann, J.L. Vujić, Algorithmic choices in WARP – a framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs, Ann. Nucl. Energy 77 (2015) 176–193.
[8] S.P. Hamilton, T.M. Evans, S.R. Slattery, GPU acceleration of history-based multigroup Monte Carlo, Trans. Am. Nucl. Soc. 115 (2016) 527–530.
[9] J. Apostolakis, R. Brun, F. Carminati, A. Gheata, S. Wenzel, A concurrent vector-based steering framework for particle transport, J. Phys. Conf. Ser. 523 (2014) 012004.
[10] J. Apostolakis, M. Bandieramonte, G. Bitzes, R. Brun, P. Canal, F. Carminati, J.C. De Fine Licht, L. Duhem, V.D. Elvira, A. Gheata, S.Y. Jun, G. Lima, M. Novak, R. Sehgal, O. Shadura, S. Wenzel, Adaptive track scheduling to optimize concurrency and vectorization in Geant V, J. Phys. Conf. Ser. 608 (2015) 012003.
[11] G. Amadio, J. Apostolakis, M. Bandieramonte, C. Bianchini, G. Bitzes, R. Brun, P. Canal, F. Carminati, J. De Fine Licht, L. Duhem, D. Elvira, A. Gheata, S.Y. Jun, G. Lima, M. Novak, M. Presbyterian, O. Shadura, R. Seghal, S. Wenzel, First experience of vectorizing electromagnetic physics models for detector simulation, J. Phys. Conf. Ser. 664 (2015) 092013.
[12] P.K. Romano, N.E. Horelik, B.R. Herman, A.G. Nelson, B. Forget, OpenMC: a state-of-the-art Monte Carlo code for research and development, Ann. Nucl. Energy 82 (2015) 90–97.
[13] N. Horelik, B. Herman, B. Forget, K. Smith, Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS), in: Int. Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, Idaho, May 5–9 2013.
[14] J.-Y. Lee, S.R. Choi, S.J. Kim, PGSFR core design and performance characteristics, Trans. Am. Nucl. Soc. 114 (2016) 700–703.
[15] N.E. Stauff, C. Lee, P.K. Romano, T.K. Kim, Verification of mixed stochastic/deterministic approach for fast and thermal reactor analysis, in: International Congress on Advances in Nuclear Power Plants, Fukui and Kyoto, Japan, Apr. 24–28 2017.
[16] J.R. Tramm, A.R. Siegel, Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes, Ann. Nucl. Energy 82 (2015) 195–202.
[17] P.K. Romano, A.R. Siegel, R.O. Rahaman, Influence of the memory subsystem on Monte Carlo code performance, in: Joint Int. Conf. Mathematics and Computation, Supercomputing in Nuclear Applications, and the Monte Carlo method, Knoxville, Tennessee, Apr. 19–23 2015.