# Continuous-Energy Monte Carlo on GPUs in Shift[1]

Steven P. Hamilton, Thomas M. Evans, and Stuart R. Slattery

*Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN*
*hamiltonsp@ornl.gov, evanstm@ornl.gov, slatterysr@ornl.gov*

## INTRODUCTION

Continuous-energy Monte Carlo offers the most accurate solutions to the radiation transport equation. However, the high computational cost associated with Monte Carlo methods often leads to the need for large-scale computing resources in order to perform simulations of physical systems of interest. To deliver high computational performance with lower energy requirements, most supercomputers have turned to GPUs or other vectorized computing architectures for the majority of their capability. For this reason, there has been interest in recent years in developing algorithms to perform Monte Carlo transport on GPUs.

Vectorized Monte Carlo was first introduced in the 1980s in response to early vector computers [1]. These methods had largely fallen out of favor until renewed interest in recent years spurred by the introduction of GPUs for scientific computing. To date, the only studies considering continuous-energy Monte Carlo neutron transport on GPUs involve the WARP code [2, 3]. While these studies offer considerable insight into the challenges associated with performing transport on GPUs, the implementation lacks many features that are necessary for a production neutron transport solver. In particular, (1) some cross section mechanisms such as $S(\alpha,\beta)$ and probability tables are not implemented, (2) all operations are performed in single- rather than double-precision arithmetic, (3) no variance reduction is implemented, and (4) only a minimal set of tallies is supported (no cell- or mesh-based tallies are available). Other recent studies have provided algorithmic comparisons for multigroup transport problems [4, 5] without considering the continuous-energy problem.

This paper describes the implementation and preliminary numerical results for continuous-energy Monte Carlo on GPUs within the Shift radiation transport code [6]. This paper expands on previous studies in the literature in several areas. To our knowledge, it is the first study in the literature describing continuous-energy neutron transport on GPUs within a production transport package. The full set of features available in the GPU solver will be described in the next section. In addition, this is the first study offering a performance comparison of CPU and GPU implementations within the same code base. Ref. [3] compared GPU performance in WARP with CPU implementations in other code bases having substantially different feature sets. Finally, this is the first study comparing history- and event-based implementations of continuous-energy transport on GPUs.

## GPU ALGORITHMS IN SHIFT

Both history-based and event-based GPU algorithms have been implemented in the Shift radiation transport code [6] using Nvidia's CUDA programming language [7]. The full set of cross section evaluations present in Shift are available, including standard tabular interpolation, $S(\alpha,\beta)$, probability tables, and on-the-fly elastic scattering. Both fixed source and $k$-eigenvalue (criticality) problem modes are available. Available geometry options are a Cartesian mesh geometry and a reactor toolkit (RTK) geometry optimized for pressurized water reactors. The standard version of Shift additionally supports SCALE and MCNP5 geometries. Support for arbitrary geometric configurations within the GPU solver using the SCALE input specification is planned. In addition to a $k$-effective tally, energy-integrated flux and reaction rate tallies are available over spatial cells (or collections of cells) as well as on an overlaid Cartesian mesh. As with the standard implementation in Shift, these tallies all use path-length estimators. Shannon entropy and spatial moment tallies can be used as diagnostics of fission source convergence for eigenvalue calculations.

One significant deviation from Shift's standard implementation is the calculation of tally statistics. The history-based variance calculation used in the CPU version of Shift requires non-trivial memory allocations for each individual particle history. Due to the large number of particles that are active at one time in the GPU algorithm, batch statistics were determined to be a more appealing approach. The OpenMC [8] and MC21 [9] codes always use batch statistics in their calculations, so the switch to batch statistics was not deemed to be a substantial sacrifice. Implicit capture (survival biasing) combined with Russian roulette is used for variance reduction. The GPU implementation shares a significant amount of code with the CPU version of Shift. Most of the pre- and post-processing steps, including all input/output and tally processing capabilities, are shared between the two solvers. Therefore, the total amount of code that had to be rewritten or duplicated to support the GPU algorithms represents a very small percentage of the overall code base in Shift.

In the history-based algorithm, each computing thread is assigned to transport a separate particle history for its entire lifetime, closely mirroring the approach used in most CPU-based Monte Carlo codes. The history-based algorithm in this study is a straightforward conversion of the CPU algorithm in Shift to run on GPUs. A few simplifications were made, such as reducing explicit caching of macroscopic and microscopic cross section data in order to reduce memory usage. These simplifications only impact the efficiency and not the accuracy

of the solver. Furthermore, differences in the execution model on the GPU indicate that optimizations that improve performance on the CPU may not be appropriate on the GPU. On the GPU, the history-based method contains a single large *kernel* (function that is executed on the GPU) that performs all aspects of the transport process. This kernel is is long-lived, as it must continue until all particles have been removed from the system. Variability in the types of operations that various particles are performing at a given time as well as variability in the total length of different particle histories leads to *thread divergence*, where threads on the same vector unit on the processor are trying to perform different operations. This thread divergence leads to reduced efficiency because only a subset of the available threads can be performing useful work at one time.

In an event-based algorithm, the transport process is performed one step at a time by collecting groups of particles performing the same type of operation or event. The event-based algorithm closely follows the approach described in Ref. [5]. In particular, three event types are considered: (1) movement of a particle to its next location (either geometric boundary or collision site), (2) processing of collisions, and (3) creation of new particles from the source. The source event type was introduced in Ref. [5] as an approach to increase occupancy on the GPU while limiting the number of particles that are active at one time. Each kernel in the history-based algorithm only performs a single operation; the kernels are therefore much smaller than in the history-based algorithm. Collecting particles into different event types before executing kernels allows for much lower thread divergence relative to the history-based algorithm. However, the process of assigning particles to different events introduces some extra computation and movement of data.

The functions that perform cross section evaluations, geometry tracking, and tallying are fully shared between the history- and event-based algorithms in Shift. Due to the small number of event types used, the total difference between the history- and event-based implementations is only a few hundred lines of code. The event-based algorithm described in Ref. [2] contained a much larger number of events and would therefore likely require more substantial code modifications in order to additionally support a history-based implementation. Similar to our event-based method, the approach described in Ref. [4] contains three event types, although the particular events are slightly different.

## RESULTS

To evaluate algorithmic performance, the simulation of a small modular reactor (SMR) core is considered. The core design is loosely based on the NuScale reactor design and contains 37 fuel assemblies of varying enrichment. Details of the geometry and materials can be found in Ref. [10]. A Cartesian mesh tally is applied with both energy-integrated flux and the fission reaction rate. The mesh for this tally contains 200 cells in each of the $x$ and $y$ directions, and 10 cells in the $z$ direction. The CPU architecture for this study is two eight-core Intel Xeon E5-2630 v3 CPUs operating at 2.4 GHz. The GPU architectures considered are the NVIDIA Tesla K40 GPU and the NVIDIA Tesla P100 GPU. Table I displays the particle tracking rate achieved using all 16 CPU cores (executing using 16 MPI tasks), as well as the history- and event-based GPU methods, on the two different GPU models. These calculations used 3.2 million neutrons per cycle for 10 cycles (5 inactive and 5 active). While the number of cycles here is insufficient to appropriately converge the fission source or produce accurate statistics, it is sufficient to illustrate the performance characteristics of the methods. On both GPUs, the performance of the event-based algorithms significantly outperforms the history-based approach. This is in contrast to the results for multigroup transport in Ref. [5], where the history-based algorithm offers far better performance than the event-based approach. The reason for this discrepancy is likely due to a variety of factors, but we posit that the primary reason is that the increased complexity of computing cross sections in the continuous-energy approach dramatically increases the amount of thread divergence. Because the event-based algorithm naturally reduces thread divergence, it is more successful in the continuous-energy case. On the P100 GPU, the event-based method achieves a particle tracking rate approximately double that of the 16 core CPU, indicating that a single GPU provides the effective capability of nearly 32 CPU cores. As expected, the particle tracking rate is slightly lower during the active cycles due to the increased cost of performing tallies. The percentage by which the tracking rate decreases during the active cycles is approximately the same on the CPU and GPU.

TABLE I. Particle tracking rate for SMR problem in thousands of neutrons per second.

| | | Tracking rate (kn/s) | |
|---|---|---|---|
| Architecture | Algorithm | Inactive | Active |
| Xeon CPU (16 core) | History | 58.4 | 50.5 |
| K40 GPU | History | 21.5 | 19.4 |
| K40 GPU | Event | 39.8 | 36.5 |
| P100 GPU | History | 59.1 | 51.9 |
| P100 GPU | Event | 115.2 | 96.8 |

Figure 1 shows the tracking rate as a function of particle count for the CPU and GPU versions of Shift for the same SMR problem. For both GPU architectures, the tracking rate for the history- and event-based algorithms is approximately the same for small particle counts. However, as the number of particles is increased, the tracking rate for the event-based method increases substantially, while the rate for the history-based method is largely flat. This indicates that a larger number of particles is necessary to saturate the GPU in the event-based method. This effect is far more pronounced on the newer P100 GPU than on the K40 GPU. It is expected that this trend will become even more severe on future GPU architectures.

## CONCLUSIONS AND FUTURE WORK

This effort implemented a GPU-enabled continuous-energy Monte Carlo particle transport solver in the Shift code using the CUDA programming language. This solver currently supports a significant number of the most commonly used features within Shift. History- and event-based algorithms are available, with initial performance testing showing that the event-based method

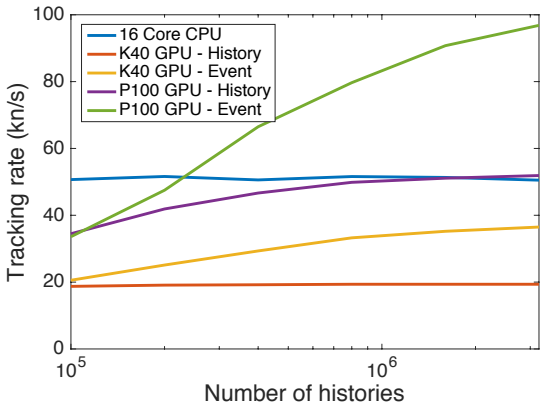


Fig. 1. Active cycle particle tracking for SMR core as a function of history count for different architectures and algorithms.

is capable of delivering nearly twice the performance of the history-based algorithm. This event-based algorithm achieves almost twice the particle tracking rate on an NVIDIA P100 GPU relative to a 16-core CPU.

A drawback to the present implementation is that larger particle counts per GPU are necessary in order to achieve peak performance. Ongoing work seeks to develop approaches to reduce the number of active histories needed to fully occupy the GPU. An additional area of ongoing research is the implementation of a general-purpose geometry package to supplement the current reactor-optimized geometry implementation. Future work will provide the domain decomposition and hybrid Monte Carlo-deterministic methods currently available in Shift.

## ACKNOWLEDGMENTS

This research was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy under Contract No. DE-AC05-00OR22725. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

## REFERENCES

1. F. BROWN and W. MARTIN, "Monte Carlo Methods for Radiation Transport Analysis on Vector Computers," *Progress in Nuclear Energy*, **14**, *3*, 269–299 (1984).
2. R. BERGMANN and J. VUJIC, "Algorithmic Choices in WARP – A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs," *Annals of Nuclear Energy*, **77**, 176–193 (2015).
3. R. BERGMANN, K. ROWLAND, N. RADNOVIĆ, R. SLAYBAUGH, and J. VUJIĆ, "Performance and accuracy of criticality calculations performed using WARP – A framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs," *Annals of Nuclear Energy*, **103**, 334–349 (2017).
4. R. C. BLEILE, P. S. BRANTLEY, M. J. O'BRIEN, and H. CHILDS, "Algorithmic Improvements for Portable Event-Based Monte Carlo Transport Using the NVIDIA Thrust Library," *Transactions of the American Nuclear Society*, **115**, 535–538 (2016).
5. S. HAMILTON, S. SLATTERY, and T. EVANS, "Multi-group Monte Carlo on GPUs: Comparison of history- and event-based algorithms," *Annals of Nuclear Energy*, **113**, 506–518 (2018).
6. T. PANDYA, S. JOHNSON, T. EVANS, G. DAVIDSON, S. HAMILTON, and A. GODFREY, "Implementation, Capabilities, and Benchmarking of Shift, a Massively Parallel Monte Carlo Radiation Transport Code," *Journal of Computational Physics*, **308**, 239–272 (2016).
7. "CUDA C Programming Guide," Tech. Rep. PG-02829-001_v7.5, NVIDIA (2015).
8. P. ROMANO and B. FORGET, "The OpenMC Monte Carlo particle transport code," *Annals of Nuclear Energy*, **51**, 274–281 (2013).
9. D. GRIESHEIMER ET AL., "MC21 v.6.0 – A continuous-energy Monte Carlo particle transport code with integrated reactor feedback capabilities," *Annals of Nuclear Energy*, **82**, 29–40 (2015).
10. P. ROMANO ET AL., "Monte Carlo full core performance baseline," Tech. Rep. ECP-SE-08-36, Exascale Computing Project (2017).