EPJ Nuclear Sci. Technol. **11**, 56 (2025)

https://doi.org/10.1051/epjn/2025046

EPJ Nuclear Sciences & Technologies

Available online at:
https://www.epj-n.org

Status and advances of Monte Carlo codes for particle transport simulation, edited by Andrea Zoia, Cheikh Diop and Cyrille de Saint Jean

REGULAR ARTICLE

OPEN ACCESS

GREAPMC: development status, capabilities, and future developments

Muhammad Rizwan Ali[1], Murat Serdar Aygul[1], and Deokjung Lee[1,2,*]

[1] Department of Nuclear Engineering, Ulsan National Institute of Science and Technology, 50 UNIST-gil, Ulsan 44919, Republic of Korea
[2] Advanced Nuclear Technology and Services, 406-21, Jonga-ro, Jung-gu, Ulsan 44429, Republic of Korea

Received: 11 June 2024 / Received in final form: 11 July 2025 / Accepted: 4 August 2025

Abstract. This article focuses on the capabilities, performance, and verification of **G**PU-optimized **REA**ctor **P**hysics **M**onte **C**arlo (GREAPMC), a multigroup Monte Carlo code, against multigroup simulation on MCS; an in-house, CPU-based, Monte Carlo code. The simulation results for a three-dimensional fuel assembly indicate a superior performance of GREAPMC over the conventional MC codes optimized for CPU with one GPU card equivalent to approximately 13 CPU boards for the hardware employed in this work. The immense potential of GPUs has led to the development of GPU-aware continuous energy capability which is in progress with the target velocity sampling (TVS) part completed. The essence of using GPU for continuous energy MC is quantified by simulating the effect of the number of histories on the execution time for the TVS methods. The results show a remarkable acceleration compared to the CPU. The findings provide a basis for further development in GREAPMC.

1 Introduction

Monte Carlo (MC) codes transport the particle within a medium using random sampling. Hence, they are subject to stochastic uncertainty [1]. Reducing the uncertainty is accomplished by increasing the number of histories/particles to be transported. However, this makes MC codes inherently computationally intensive. The demand for faster, yet accurate simulations translates to using cluster computing which can skew the relation between the speedup and efficiency [2]. A general-purpose graphical processing unit (GPU) with thousands of cores is ideal for the MC method which is generally considered to be embarrassingly parallel [3].

Graphics Processing Units (GPUs) have evolved beyond their original role in computer graphics. Unlike Central Processing Units (CPUs) with a few high-performance cores, GPUs boast thousands of simpler cores optimized for parallel execution. These cores are grouped to form streaming multiprocessors (SMs). A warp, consisting of 32 threads, forms a thread block and each thread block is executed on an SM.

The number of threads programmed in an application can be much larger than the number of threads that can run on an SM simultaneously. To address this, GPUs employ sophisticated schedulers. These schedulers dynamically manage the loading and unloading of thread blocks onto SMs. They consider factors like thread completion status and memory access patterns to optimize resource utilization. By efficiently swapping between thread blocks, schedulers can hide memory latency and maintain high throughput despite the vast number of threads.

GPUs employ a memory hierarchy tailored for parallel processing. Global memory, the largest but slowest type, stores data accessible by all threads. However, its limited bandwidth can hinder performance. To mitigate this, GPUs uses caches and shared memory. Shared memory, local to an SM, offers faster access for frequently used data within a thread block. L1 and L2 caches further reduce memory access latency by storing frequently accessed global memory data closer to the processing cores. Finally, thread-private registers provide the fastest storage but are limited in size. Efficient memory management and data transfer between these memory levels are crucial for optimal GPU performance.

The reactor physics community is already employing GPUs for MC development. Bleile et al. sought algorithmic improvements for the MC method, specifically focusing on the event-based method, using the Thrust library [4]. They quantified the effect of using Structure of Arrays (SOA) instead of Array of Structures (AOS) for the particle data. Okubo et al. performed multi-group calculations

* e-mail: deokjung@unist.ac.kr

using delta-tracking on GPUs and verified the method against benchmark results. The ratio of calculation time between CPU and GPU was found to be around 7 [5]. Liu et al. compared the performance of GPU against Intel CPU and found a speedup factor of 2.2 [6]. Another optimization study on event-based neutron tracking, utilizing a multigroup MC code called Quicksilver [7], employed a queue-driven methodology [8] for improved efficiency. In this work, the history-based algorithm achieved superior performance for multigroup material treatment, and the event-based algorithm was superior for continuous energy material treatment. Gao et al. enumerate the development of a GPU-accelerated MC code MagiC using the event-based methodology and energy sorting of the cross-section data [9]. They found an impressive acceleration of around 2.5 compared to the AMD 7371 CPU for the Hoogenboom-Martin performance benchmark problem [10]. Recently, Lawrence Livermore National Laboratory (LLNL), USA reported their work on porting the MC codes to the Sierra GPU [11,12]. The optimizations have resulted in a speedup of 7.61 for neutronic problems [12]. WARP, the pioneering continuous energy code designed explicitly for GPUs, implements event-based neutron tracking [13] and utilizes the GPU-accelerated ray tracing framework OptiX for geometry representation [14]. Another notable GPU-specific code, GUARDYAN, specializes in time-dependent MC calculations for nuclear transients [15]. OpenMC has also been adapted to use GPU processing power [16], with a significant portion of its functionalities successfully migrated to the GPU code base [17]. Shift and PRAGMA are two continuous-energy MC codes, with PRAGMA exclusively developed for GPUs [18], while Shift has been adeptly modified to operate on both CPUs and GPUs [19]. Notably, Shift's GPU adaptation was preceded by comprehensive investigations conducted on Profugus [20], which leveraged multigroup cross-sections. Profugus presented several novel optimizations for history-based and event-based neutron tracking. In the history-based regime, a new method was presented where the threads that have completed the simulation of the particles are provided with new data periodically. This method was included in PRAGMA with further refinement. For multigroup cross-sections, history-based tracking offers better performance because it uses fewer registers, whereas event-based tracking suffers from performance degradation due to the need for particle data sorting between events [20]. Hamilton et al. reported superior performance of Shift with history-based neutron tracking using continuous energy cross-section for the fresh fuel and with event-based neutron tracking for the depleted fuel [19]. The same trend was reported by Namjae et al. [18]. Particles are sorted between the events in an event-based neutron tracking method. Cuneo et al. provided a new on-GPU asynchronous scheduling method to reduce thread divergence [21]. The application of the method to MC particle transport can result in further acceleration of the tracking algorithm due to reduced thread divergence. In a recent work, Morgan et al. explored the production options for rapid prototyping and testing novel methods with GPUs [22]. They tested PyKokkos [23], Numba threading [24], and Numba PyOMP [25] and found pyKokkos to be more performant. However, pyKokkos requires a large translation and compilation time [22].

Keeping in view the requirement of analysis and design of Pressurized Water Reactors (PWRs), the UNIST CORE lab is developing GREAPMC (**G**PU-optimized **REA**ctor **P**hysics **M**onte **C**arlo); a GPU-accelerated Monte Carlo code [26,27] using object-oriented programming (OOP) paradigm in the CUDA C++ programming model. The aim of development is to perform PWR simulations on GREAPMC, employing GPUs for a reduction in execution time, while the remaining research work shall still be performed using MCS, a general-purpose high-fidelity in-house MC code [28]. Hence, the timing comparison and verification are performed solely against MCS.

GREAPMC is still under development with the optimization of history-based neutron tracking and geometrical treatment already completed. Particle tracking in GREAPMC relies on conventional surface tracking with history-based neutron tracking for particle transport. Currently, the target is optimizing the geometry and tracking algorithm for the fresh fuel modeling, hence history-based neutron tracking is selected for particle transport. It is pertinent to mention that most of the particle data in GREAPMC uses single-precision storage in contrast to MCS which uses double precision throughout. This article elaborates on the benefits and shortcomings of GREAPMC's algorithmic choices, specifically focusing on the benefits of using GPUs for MC codes. The acceleration is compared with MCS, which is run in multi-group mode to ensure a consistent basis for comparison.

Using multigroup cross-sections limits the accuracy of material representation in GREAPMC, necessitating a shift towards continuous-energy cross-sections. To enhance GREAPMC's applicability, the CORE lab is actively developing the capability of GPU-accelerated continuous-energy treatment, with the development of the target velocity sampling (TVS) module complete. This article also presents the verification of TVS results against MCS under same simulation settings, along with the comparison of acceleration. This continuous-energy module is designed for later integration with GREAPMC. Afterward, the event-based neutron tracking shall be incorporated into the code before moving on to depletion.

The remainder of the article is divided into four sections. The second section describes the current status of multi-group GREAPMC. The third section then presents the results and explores the performance implications of the algorithmic choices. Subsequently, the fourth section details the brief background of the target velocity sampling treatments. Adhering to the aim of developing a simulation tool that can execute problems in less time than MCS, the target velocity sampling algorithms are verified against MCS in the same section. Finally, the fifth section concludes and provides a preview of future development goals for the code.

2 Capabilities

In the interest of facilitating a comprehensive understanding of the code's potential, this section details

GREAPMC's current capabilities, including optimization of history-based neutron tracking, cell-based geometry modeling, and cell-based tally.

2.1 History-based neutron tracking

GREAPMC optimizes history-based neutron tracking by reducing each thread's idle time. Conventional history-based neutron tracking is where a particle is traced from birth to termination within one monolithic transport loop. The inherent variability in the history lengths causes idle threads leading to loss of the computational resources on GPUs. Hamilton et al. presented a modified history-based neutron tracking [20]. The same was later refined and incorporated into PRAGMA [18]. This GPU-optimized tracking algorithm is termed history-length truncation [20] and it is implemented in GREAPMC as a secondary method. In this method, for efficient resource utilization, the transport loop is controlled such that it iterates until the particle undergoes a specific number of interactions. Afterward, the particles are sorted in the host side (CPU) according to the dead/alive state, and the GPU kernel is again invoked with only alive particles and the process continues until all the particles are marked dead. This method aims at increasing resource utilization by reducing the number of idle threads. The number of interactions before the next sorting operation is a hyper-parameter and depends on the problem configuration and material information. Another optimization that is originally applied in event-based neutron tracking in Profugus (source event method) [20] has been refined for history-based neutron tracking. This algorithm is coined as the *history-replenishment method* and serves as the primary history-based neutron tracking method in GREAPMC. Like the source event method, history-replenishment assigns new particles from the same particle data to the threads that have completed simulating the particles. In this way, threads are always busy simulating the particles. The difference lies in the application and the number of threads employed. In history-replenishment, the kernel is always invoked with K threads where $K \leq N/2$, $N$is the number of histories. Reducing K up to a limit will increase the performance. The optimal value of K depends again on the problem specifications and can potentially be optimized using machine learning. Source event method employed $N/10$ threads [20] while in GREAPMC, the value is empirically chosen as $N/60$.

2.2 Geometry modeling

While the majority of MC codes employ constructive solid geometry (CSG) [28–31], GREAPMC uses cell-based geometrical modeling instead of CSG. The benefit of cell-based modeling is that the cell where a particle is located is known throughout the transport loop. CSG can potentially lead to the variable execution time of each thread owing to the coordinate level a particle resides in. Additionally, the particle location is quantified multiple times within the transport loop, although this operation incurs a minimal overhead as the codes maintain a neighbor list for each cell. This list contains the identifiers for neighboring cells. In most cases, this search is enough to locate the particle in the CSG-based model. Nevertheless, replacing CSG with cell-based modeling can reduce the slight variability of execution time during the search for a particle's location. Now the search is performed only once before the particle transport loop begins. The downside of cell-based geometrical modeling is the loss of generality in modeling different types of reactors. However, as the focus of the code is only modeling PWRs now, cell-based geometrical modeling is deemed suitable.

2.3 Cell-based tally operation

Tally bin index searching is among the most crucial aspects when considering the computational burden of the MC method. Typically, MC codes employ a hash function to accelerate the mapping between the particle's location and the index into the tally array [32,33]. MCS employs a very efficient hash function which shows a superior performance compared to other high-fidelity MC codes [6]. The hash function of MCS is taken as a guide to design the tally bin index searching scheme for GREAPMC.

The functionality of mesh tally has not been implemented in GREAPMC as the region/call-based tally performs adequately well even if the number of cells is increased [27]. Conversely, traditional CSG-based MC codes like MCS experience performance degradation in cell-based tallies mainly due to the overhead associated with surface tracking.

However, one potential limitation of GPU-based MC codes is using atomic add operation in tallies. An atomic add operation is guaranteed to be performed without interference from other threads. Consequently, with a significant increase in the quantities to be tallied, atomic add operations can become a bottleneck, hindering performance.

3 Performance quantification

All the simulations in this section are run on a 3D assembly with black boundary conditions axially while other boundaries are considered reflective. Figure 1 shows the fuel assembly model. The material macroscopic cross-sections and the geometric details are adopted from the C5G7 benchmark [34]. The axial geometry is modified such that the dimensions of the assembly are $21.42 \times 21.42 \times 380\,\text{cm}$. The assembly contains a 10 cm moderator region at the top and the bottom respectively, and the active fuel region is divided into 50 uniform regions. The GREAPMC results are quantified against MCS with five million particles per cycle, 70 inactive and 300 active cycles for all the simulations. The same multi-group cross-sections are used in the inputs of both the codes: MCS and GREAPMC. GREAPMC simulations are performed on one NVIDIA GeForce RTX 3090 GPU card while MCS simulations use 60 threads of AMD EPYC 7452 which has a base clock of 2.35 GHz and 32 cores

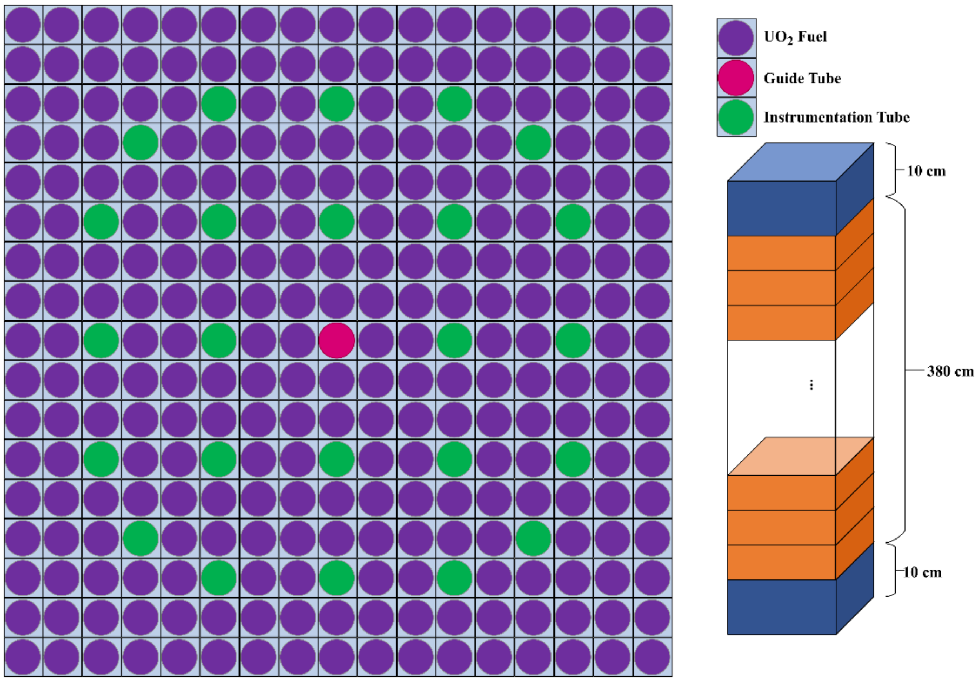

Fig. 1. The fuel assembly model used in simulations.

with 2 threads per core. The remaining four threads were reserved for debugging, development, and system tasks.

The performance of GREAPMC with an increase in axial regions has already been quantified [27]. In such a case, GREAPMC incurs a minimum increase in execution time with the number of axial regions. The effect arises due to differences in the computing architecture and because the cell is always known, eliminating the need to even search the neighbor list. In the current work, the impact of surface crossing on the execution time is quantified by increasing the number of rings from one to six. For these six cases, the speedup in terms of the equivalent CPU core count is calculated using equation 1. This equation compares the single CPU core performance against single GPU.

$$\text{Equivalent CPU Cores} = \frac{(\text{Number of CPU cores}) \times (\text{CPU run time})}{\text{GPUrun time}}. \quad (1)$$

Figure 2 illustrates the equivalent CPU cores required to match the performance of a single GPU card for simulations with varying numbers of rings. History replenishment is employed for these results. The plot demonstrates an increasing trend in CPU cores with increasing rings, suggesting that the necessary core count would continue to rise for more complex geometries. Notably, the equivalent CPU cores reach 845 at six rings. This signifies that, for a six-ring simulation, MCS would require 845 CPU cores to achieve the same execution time as a single GPU card running GREAPMC.

The current GREAPMC version uses multigroup material treatment. With continuous energy cross-sections, global memory accesses are much larger than in the multigroup case and act as bottlenecks. Nevertheless, current development aims to optimize history-based neutron tracking to simulate fresh fuel cases. The performance of GREAPMC is not solely due to the geometric treatment. CSG-based MC codes maintain a neighbor list to search for the particle; hence, the subroutine to locate a particle's cell readily quantifies the particle's cell. The performance disparity between GREAPMC and MCS is primarily attributed to differences in the underlying architecture and execution model of the GPU and CPU. Cell-based geometry only eliminates the function calls to find the cells. The performance of history replenishment is equivalent to the history length truncation method. For instance, with the specific number of interactions after which the particles are sorted in the history length truncation method set to an empirical value of 40, the six-ring case provides an equivalent CPU count of 843. This means both methods incur nearly identical execution times when fully optimized, for the multigroup material representation. Profugus also used a multigroup material representation [20], however, we cannot compare our results with Profugus's as the GPU model used in their work is different. The number of CUDA cores and memory size impact the performance.

The number of surface crossings in surface tracking is expected to increase as geometry includes more radial rings. The rise in surface crossings is attributed, in part, to the reflective boundary condition. With the reflective boundary condition, there is no loss of neutrons via leakage hence more neutrons are available for surface crossings.

Loss of performance encountered by adding the radial rings to the fuel can be quantified using a ratio between execution time with one ring to any number of rings as

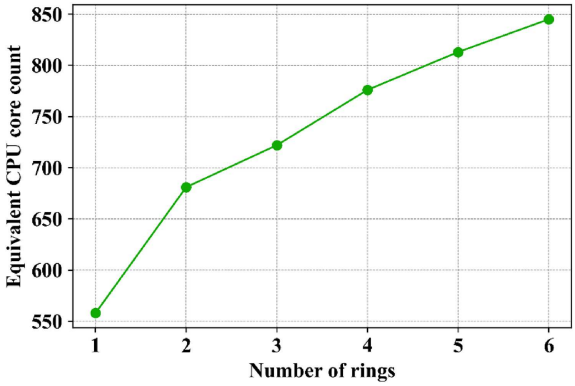

Fig. 2. Equivalent CPU core count versus the number of rings in the fuel rod.

Table 1. The loss percent signifies the increase in execution time due to the radial rings.

Radial rings in fuel rod	MCS (%)	GREAPMC (%)
2	28	13
3	41	24
4	50	31
5	57	38
6	63	44

given in equation 2.

$$\text{Loss percent} = \left(1 - \frac{\text{Execution time with one ring}}{\text{Execution time with } n \text{ rings}}\right) \times 100. \quad (2)$$

Table 1 shows the values of the loss percent with MCS and GREAPMC. The ideal value of the loss percent is zero signifying no performance penalty with an increase in radial rings. As more rings are added, both codes suffer the performance penalty owing to more surface crossings. However, the performance loss with MCS is greater than GREAPMC. With 6 rings, GREAPMC runs 44% slower than the case with one ring while MCS suffers a slowdown of 63%. MCS performance loss also includes the communication overhead due to the MPI calls.

Delta tracking [35] addresses the performance degradation due to a significant increase in surface crossing events when modeling fuel rods with multiple radial rings. It effectively treats the fuel rod as a single ring in the radial direction, minimizing surface crossings and enhancing computational efficiency.

To mitigate the potential issue of heavy absorbers or high levels of material heterogeneity, a modified multi-regional delta tracking method [36] can be adopted where delta tracking is confined to the fuel region itself.

The assessment of the tracking rate per unit cost (neutrons/s/$) gives further insight into the benefits of using GPU for the MC method. Taking the cost estimate of 60 CPU cores and RTX 3090 GPU card, the tracking rates are divided by the unit costs to obtain Figure 3. The cost

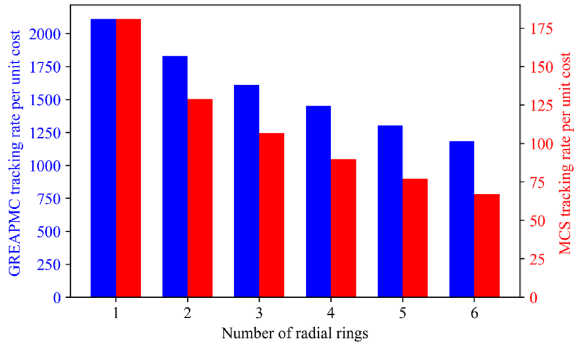

Fig. 3. Comparison of tracking rate per unit cost between MCS and GREAPMC.

of a GPU card is estimated at $1600, while the price of 60 CPU processors is roughly taken as $2000. The left y-axis and blue bars quantify the tracking rate per unit cost for GREAPMC while the right y-axis and red bars quantify the MCS values. The tracking rate of both codes decreases as the number of rings increases.

GREAPMC achieves a tracking rate per unit cost of around 1180 for simulations with six radial rings. In contrast, MCS manages a tracking rate per unit cost of approximately 67. This significant difference demonstrates the potential economic efficiency of GPUs for running large-scale Monte Carlo simulations.

4 Target velocity sampling

TVS is a crucial factor for accurately modeling thermal and epithermal neutron energies. The thermal motion of target nuclides significantly influences the elastic scattering cross-section, especially for heavy nuclides with prominent resonance regions, such as ^{238}U. Scattering interactions with these nuclides can lead to neutron up-scattering. This up-scattering must be carefully considered as it can result in an over-estimation of reactivity [37]. Therefore, proper quantification of TVS is essential for the accurate analysis and safe operation of light water reactors (LWRs).

Among the various methods developed to account for the energy dependence of the elastic scattering cross-section while sampling target velocities, we have implemented four such methods; constant cross-section (CXS) method [38], doppler broadening rejection correction (DBRC) [39], relative velocity sampling (RVS) [38], and fast effective scattering kernel (FESK) [37]. Similar to the Shift code [19], the neutron energy within the energy grid is searched using a hash function with 8000 lethargy bins [40]. Although Relative Speed Tabulation (RST) is developed specifically for GPUs by Namjae et al. [41], the method is not considered due to the memory overhead associated with the storage of tabulated data. The methods discussed briefly here are widely used in the reactor physics community and are implemented in almost all GPU-based Monte Carlo codes [17].

The CXS method, also known as *sampling of the velocity of the target nucleus* (SVT) [42] assumes that the

cross-section remains relatively constant within the range of relative target velocities. This approximation is valid for the majority of nuclides but can significantly impact up-scattering incidents for nuclides having strong resonance regions, introducing distortions in eigenvalue calculations and fuel temperature coefficient [37].

The DBRC method is one of the approaches to address the shortcomings of CXS. This method involves determining the maximum elastic cross-section at zero Kelvin temperature within the range of relative velocities. After performing relative velocity sampling using the CXS method for a given temperature, the DBRC method applies a rejection step by comparing the sampled cross-section to the maximum elastic scattering cross-section in that region. This process is repeated until an acceptable sample is obtained. However, the DBRC method can be computationally demanding due to the numerous rejections, particularly in regions with strong resonances.

The primary distinction of the RVS method from the DBRC method is that instead of sampling relative velocity, the RVS method directly samples the target velocity using a cumulative distribution function (CDF) derived from the zero Kelvin temperature cross-section. This approach significantly reduces the number of rejections compared to the DBRC method.

The implementation of FESK, also known as the *weight correction method* (WCM) [43], is straightforward. It samples the relative velocity using the CXS method. The pdf for the angle is assumed isotropic while velocity is sampled from a Maxwell-Boltzmann distribution. This method provides the accuracy equivalent to the DBRC method with an execution time similar to the CXS method. The only downside is that it can affect the statistics of the tallies due to the large weight variation of the neutrons. Hence, it requires an appropriate variance reduction technique to improve the statistics.

Since FESK directly operates on the neutron's weight, its performance cannot be quantified unless the continuous-energy cross-section treatment code is coupled to GREAPMC. Consequently, in this work, the performance of CXS, DBRC, and RVS on GPU is compared to the performance governed by MCS on CPU.

4.1 Verification of elastic scattering treatment

GPU-accelerated elastic scattering techniques are verified for two low-energy resonances of ^{238}U: 6.67 eV and 36.67 eV, at a temperature of 900 K. The resonances are shown in Figure 4. Neutrons with slightly lower energies (6.52 eV and 36.25 eV, respectively) are sampled using three methods: CXS, DBRC, and RVS. The number of particles is varied from five to ten million to assess the performance. Here, the performance is assessed on a single CPU core against a single GPU card. Jensen–Shannon divergence (JSD) [44] is employed to quantitatively compare the probability distributions obtained from GPU-accelerated TVS code and MCS. MCS does not employ RVS, therefore, RVS results are compared to DBRC results from MCS. The JSD score is a metric used

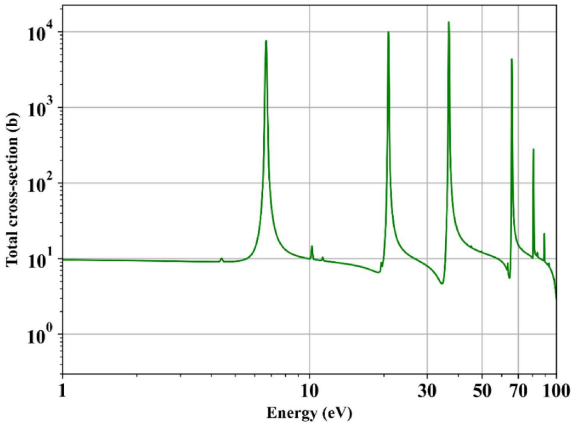

Fig. 4. ^{238}U total cross section at low energies.

to quantify the similarity between two probability distributions. It combines the information from the Kullback-Leibler divergences in both directions, providing a single measure of dissimilarity. The formula for Jensen-Shannon divergence is provided in equation 3.

$$JSD\left(P||Q\right) = \frac{K\left(P||M\right) + K\left(Q||M\right)}{2}. \tag{3}$$

Here K denotes the Kullback-Leibler divergence and M is the pointwise mean of P and Q. A smaller JSD value signifies a higher degree of similarity between the two probability distributions.

4.1.1 6.52 eV scattering kernel

Figure 5 to Figure 7 present the probability distributions obtained using the CXS, DBRC, and RVS methods, respectively, for a 6.52 eV neutron. Since the GPU implementation of CXS and DBRC leverages code from MCS, we expect a good agreement between the results from both approaches. This expectation is confirmed by Figure 5 and Figure 6, which indeed show a close qualitative match between the distributions generated by the GPU code and the MCS for both CXS and DBRC. Similarly, the RVS probability distribution from the GPU code matches the DBRC distribution from MCS.

Table 2 provides a quantitative assessment of the agreement between the probability distributions generated by MCS and the GPU code for three different TVS methods: CXS, DBRC, and RVS. The table shows the JSD scores obtained for simulations involving 10 million simulated particles. JSD values in the order of 10^{-5} are typically interpreted as an excellent agreement between the probability distributions generated by MCS and GPU code.

Figure 8 illustrates the execution time of the three methods implemented in GPU code as the number of particles increases. As expected, CXS incurs minimal execution time due to its straightforward implementation. However, DBRC experiences the most significant increase in execution time owing to the rejections. Rejections can lead to thread divergence where some threads become idle

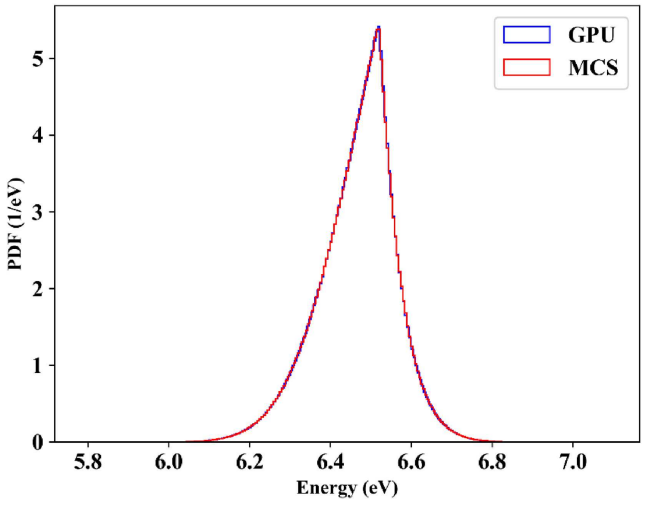

Fig. 5. CXS 6.52 eV scattering kernel with MCS and GPU code.

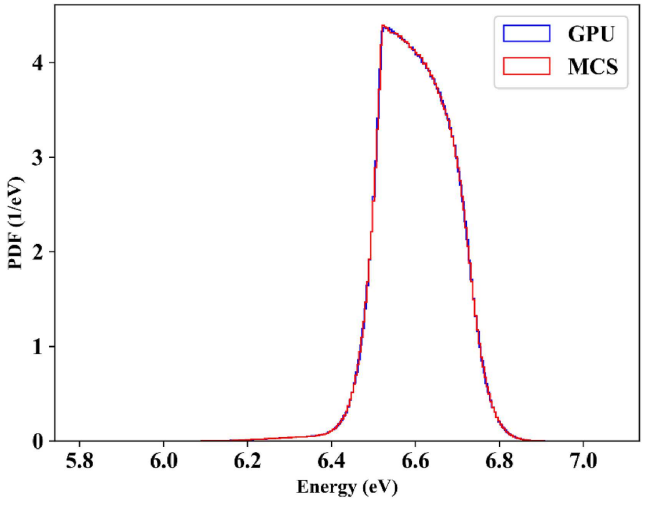

Fig. 6. 6.52 eV scattering kernel of ^{238}U with DBRC.

while waiting for others to complete calculations, impacting overall performance. Notably, DBRC has a higher number of rejections compared to RVS, resulting in its longer execution time.

The ratio of CPU time required by MCS to the GPU execution time for both CXS and DBRC methods are calculated. We observe that DBRC achieves a speedup of 188 compared to CPU execution, while CXS achieves a speedup of 107. This indicates that both methods benefit significantly from GPU acceleration.

4.1.2 36.25 eV scattering kernel

Figure 9 to Figure 11 present the probability distributions obtained using the CXS, DBRC, and RVS methods, respectively, for the scattering kernel associated with 36.25 eV. These figures serve to visually confirm the excellent qualitative agreement observed between the results from the GPU code and MCS. The excellent qualitative agreement between the GPU code and MCS provides strong validation for the effectiveness of the GPU implementation. This successful replication of the MCS behavior across different TVS methods (CXS, DBRC, and RVS) strengthens the confidence in developing the GPU-accelerated continuous energy treatment code.

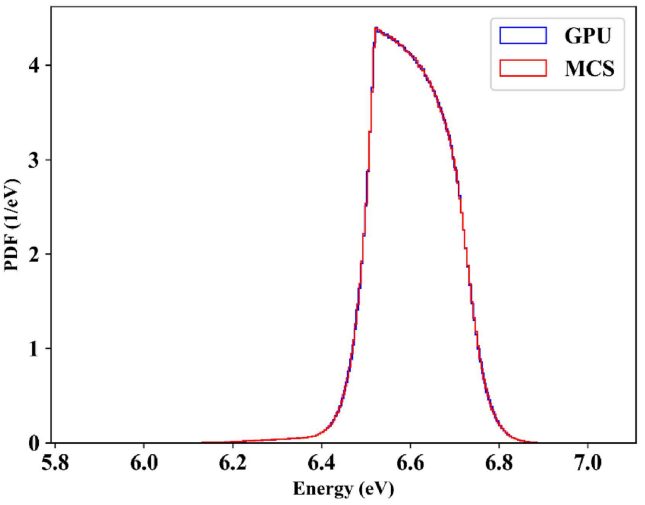

Fig. 7. 6.52 eV scattering kernel using RVS from GPU code plotted with DBRC from MCS.

Table 2. JSD values for scattering kernels at 6.52 eV.

Method	JSD
CXS	5.61×10^{-5}
DBRC	4.08×10^{-5}
RVS	4.08×10^{-5}

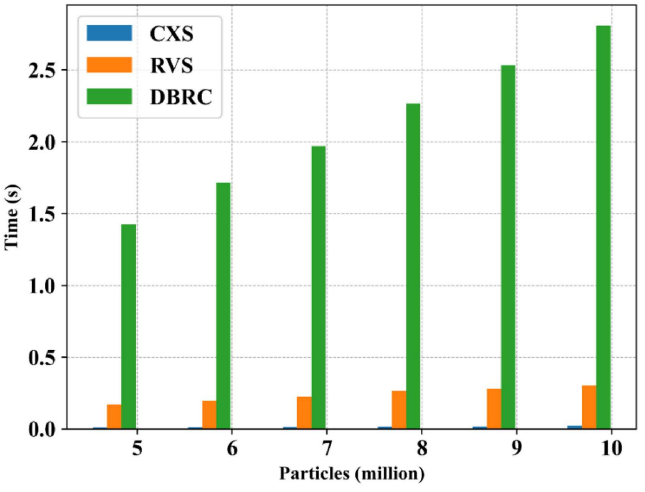

Fig. 8. Execution time of TVS methods using GPU code for 6.52 eV scattering kernel.

Table 3 presents the JSD scores at an energy of 36.25 eV, derived from simulations conducted with 10 million particles using the CXS, DBRC, and RVS methods. The consistently low JSD scores observed across all methods demonstrate a high degree of concordance between the

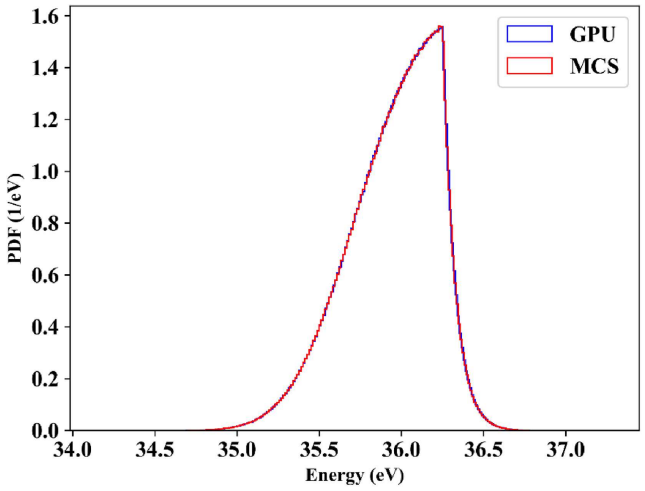

Fig. 9. 36.25 eV scattering kernel with CXS method.

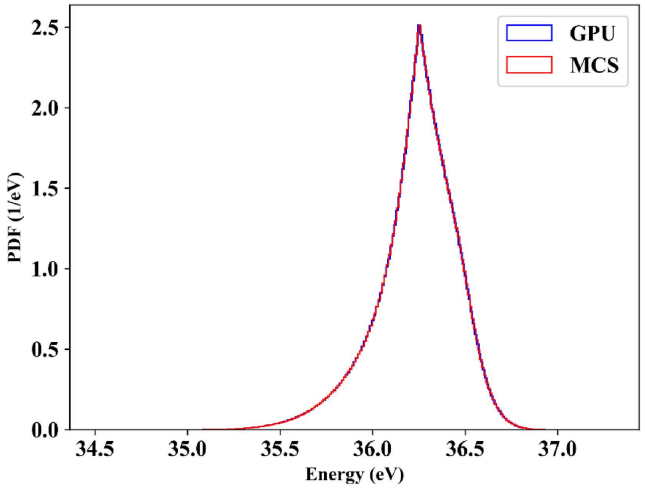

Fig. 10. Scattering kernel at 36.25 eV using DBRC.

probability distributions generated by MCS and the GPU code.

Figure 12 illustrates the execution time as a function of the number of particles (in millions) for the three TVS methods. In comparison to Figure 8, which focuses on the 6.52 eV scattering kernel, the DBRC method exhibits a nearly twofold increase in execution time, whereas the CXS method's execution time remains relatively constant. The RVS method also shows an increase in execution time, although it is less pronounced than that of the DBRC method. This disparity can be attributed to the higher rate of rejections encountered at the 36.25 eV scattering kernel at 900 K compared to the 6.52 eV case, significantly affecting the performance of the DBRC method.

The ratio of CPU time required by MCS to the GPU execution time remains noteworthy, with a factor of 462 for DBRC and 112 for CXS. This demonstrates the substantial efficiency gains achieved through GPU acceleration.

Despite the observed increases in execution time for all methods as the number of particles increases, Figure 8 and

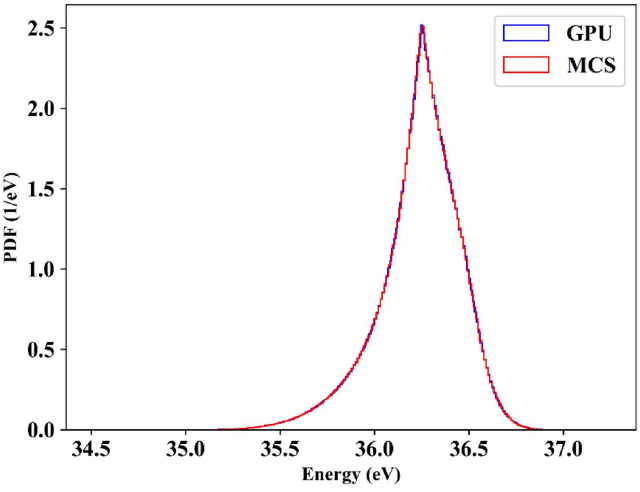

Fig. 11. ^{238}U scattering kernel at 36.25 eV using RVS from GPU code and DBRC from MCS.

Table 3. JSD scores at 36.25 eV for scattering kernel obtained via CXS, DBRC, and RVS.

Method	JSD
CXS	1.50×10^{-5}
DBRC	9.21×10^{6}
RVS	9.20×10^{6}

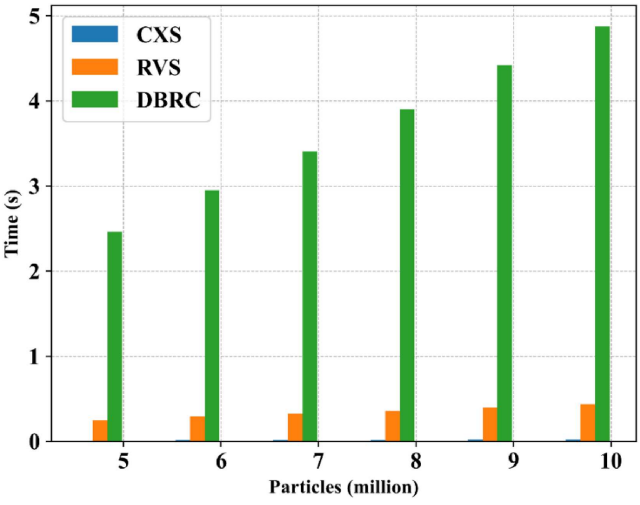

Fig. 12. Execution time of TVS methods on GPU for 36.25 eV scattering kernel.

Figure 12 underscore the substantial benefits of utilizing GPU acceleration. These figures reaffirm the significant advantage of the GPU code and the necessity of using GPUs for running large-scale MC simulations.

5 Conclusions and future directions

This work has demonstrated the significant potential of GPU-accelerated computing for Monte Carlo (MC) simulations in reactor physics. We introduced GREAPMC, an in-house GPU-optimized multigroup MC code. The current work aims to replace MCS with GREAPMC for PWR simulations. Hence, all the comparisons in this article are against MCS. Our findings reveal that for a 3D assembly case GREAPMC on a single GPU card offers equivalent performance to approximately 840 CPU cores employed in this work, highlighting the performance gains achievable with GPUs. However, we also found that an increase in the surface crossing events detriment the performance when surface tracking is employed. In this context, delta tracking shall be implemented soon.

Recognizing that a multigroup MC code has limited capabilities, we are actively developing a GPU-aware continuous-energy code. The target velocity sampling (TVS) component of this code is complete, and verification results show an impressive acceleration compared to MCS. Three TVS methods are verified namely constant cross-section (CXS), doppler broadening rejection correction (DBRC), and relative velocity sampling (RVS). The similarity in the probability distributions obtained via GREAPMC and MCS for these methods is quantified using Jensen–Shannon divergence (JSD). MCS employs double precision while the GPU code is developed to use single precision. The difference in the precision is reflected in the JSD values. However, JSD values remain in the range of 10^{-6}–10^{-5}. A fourth method, fast effective scattering kernel (FESK), has also been implemented in the code. However, its execution requires coupling GREAPMC with a GPU-aware continuous-energy treatment code, which we anticipate completing soon. Among the three methods discussed here, RVS is the most suitable method for real-time simulations on the GPU architecture due to its minimal execution time and accurate results.

The inclusion of continuous-energy cross-section treatment into GREAPMC will result in a large increase in register use, global memory accesses, and kernel size itself. Hence, we plan to implement event-based tracking to reduce the register pressure and increase coalescing in the global memory accesses.

Funding

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT). (No. NRF-2019M2D2A1A03058371) [50%] and by Korea Hydro & Nuclear Power Co., ltd (No. 2022-Tech-13) [50%].

Conflicts of interest

The authors declare that they have no competing interests to report.

Data availability statement

No data are associated with this article.

Author contribution statement

Muhammad Rizwan Ali: manuscript preparation, interpretation of results, graphs, main developer, Murat Serdar Aygul: simulations, collection of results, programming, revision, Deokjung Lee: research group leader, final manuscript revision, conceptualization, funding.

References

1. A. Haghighat, *Monte Carlo Methods for Particle Transport* (CRC Press, USA, 2014)
2. J.L. Gustafson, Amdahl's Law, in *Encyclopedia of Parallel Computing.* (Springer, USA, 2011)
3. J.E. Hoogenboom, Is Monte Carlo embarrassingly parallel?, in *Internation Conference on Physics of Reactors (PHYSOR 2012): Advances in Reactor Physics - Linking Research, Industry, and Education*, (American Nuclear Society (ANS), Knoxville, TN, USA, 2012)
4. R.C. Bleile, P. Brantley, M. O'Brien, H. Childs, Algorithmic improvements for portable event-based Monte Carlo transport using the Nvidia thrust library, in *Transactions of the American Nuclear Society*, (American Nuclear Society (ANS), Las Vegas, NV, USA, 2016)
5. T. Okubo, T. Endo, A. Yamamoto, An efficient execution of Monte Carlo simulation based on delta-tracking method using GPUs, Nucl. Sci. Eng. **54**, 30 (2017)
6. T. Liu, N. Wolfe, C.D. Carothers, W. Ji, X.G. Xu, Optimizing the Monte Carlo Neutron Cross-Section Construction Code XSBench for MIC and GPU Platforms, Nucl. Sci. Eng., **185**, 232 (2017)
7. D. Ma, B. Yang, Q. Zhang, J. Liu, T. Li, Evaluation of Single-Node Performance of Parallel Algorithms for Multigroup Monte Carlo Particle Transport Methods, Front. Energy Res. **9**, 17 (2021)
8. P.K. Romano, A.R. Siegel, Limits on the efficiency of event-based algorithms for Monte Carlo neutron transport, Nucl. Eng. Technol. **49**, 1165 (2017)
9. K. Gao, Z. Chen, A. Sun, T. Yu. Development and verification of the GPU-based Monte Carlo particle transport program MagiC, in *5th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, (Institute of Electrical and Electronics Engineers (IEEE), China, 2023)
10. 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 (M&C)*, (American Nuclear Society (ANS), Rio de Janeiro, Brazil, 2011)
11. M. McKinley, R.C. Bleile, P. Brantley, S. Dawson, et al., Status of LLNL Monte Carlo transport codes on Sierra GPUs, in *International Conference on Mathematics and Computation (M&C)*, (American Nuclear Society (ANS), Oregon, USA, 2019)
12. M. Pozlup, R.C. Bleile, P. Brantley, S. Dawson, et al., Progress porting LLNL Monte Carlo transport codes to Nvidia GPUs, in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C)*, (American Nuclear Society (ANS), Ontario, Canada, 2023)
13. F.B. Brown, W.R. Martin, Monte Carlo methods for radiation transport analysis on vector computers, Prog. Nucl. Energy, **14**, 269 (1984)

14. 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**, 176 (2015)
15. B. Molnar, G. Tolnai, D. Legrady, A GPU-based direct Monte Carlo simulation of time dependence in nuclear reactors, Ann. Nucl. Energy **132**, 46 (2019)
16. G. Ridley, B. Forget, Design and optimization of GPU capabilities in OpenMC, in *ANS Winter Meeting*, (American Nuclear Society (ANS), USA, 2021)
17. J.R. Tramm et al., Toward portable GPU acceleration of the OpenMC Monte Carlo particle transport code, in *International Conference on Physics of Reactors (PHYSOR)*, (American Nuclear Society (ANS), USA, 2022)
18. N. Choi, K.M. Kim, H.G. Joo, Optimization of neutron tracking algorithms for GPU-based continuous energy Monte Carlo calculation, Ann. Nucl. Energy **162**, 108508 (2021)
19. S.P. Hamilton, T.M. Evans, Continuous-energy Monte Carlo neutron transport on GPUs in the shift code, Ann. Nucl. Energy **128**, 236 (2019)
20. S.P. Hamilton, S.R. Slattery, T.M. Evans, Multigroup Monte Carlo on GPUs: comparison of history and event-based algorithms, Ann. Nucl. Energy **113**, 506 (2018)
21. B. Cuneo, M. Bailey, Divergence reduction in Monte Carlo neutron transport with on-GPU asynchronous scheduling, ACM Trans. Model. Comput. Simul. **34**, 1 (2024)
22. J.P. Morgan, T. Palmer, K.E. Niemeyer, Explorations of python-based automatic hardware code generation for neutron transport applications, in *Transactions of The American Nuclear Society*, (American Nuclear Society (ANS), USA, 2022)
23. N.A. Awar, S. Zhu, G. Biros, M. Gligoric, A performance portability framework for Python, in *Proceedings of the 35^{th} ACM International Conference on Supercomputing*, (Association of Computing Machinery (ACM), NY, USA, 2021)
24. S.K. Lam, A. Pitrou, S. Seibert, Numba: a LLVM-based Python JIT compiler, in *Proceedings of the seocnd workshop on LLVM compiler Infrastructure in HPC*, (Association of Computing Machinery (ACM), NY, USA, 2015)
25. T.G. Mattson, T.A. Anderson, G. Georgakoudis, PyOMP: multithreaded parallel programming in python, Comput. Sci. Eng. **23**, 77 (2021)
26. M.R. Ali, M.S. Aygul, D. Lee, GPU-optimized Monte Carlo code development - preliminary results, in *Proceedings of the Reactor Physics Asia (RPHA)*, (Korean Nuclear Society (KNS), Gyeongju, Korea, 2023)
27. M.R. Ali, M.S. Aygul, D. Lee, Enhancing PWR Monte Carlo simulations with GREAPMC: a GPU-accelerated approach, in *Physics of Reactors (PHYSOR)*, (CA, USA, 2024)
28. H. Lee et al., MCS – A Monte Carlo particle transport code for large-scale power reactor analysis, Ann. Nucl. Energy **139**, 107276 (2020)
29. P.K. Romano et al., OpenMC: a state-of-the-art Monte Carlo code for research and development, Ann. Nucl. Energy **82**, 90 (2015)
30. J. Leppänen et al., The serpent Monte Carlo code: status, development and applications in 2013, Ann. Nucl. Energy **82**, 142 (2015)
31. K. Wang et al., RMC – a Monte Carlo code for reactor core analysis, Ann. Nucl. Energy **82**, 121 (2015)
32. K. Li et al., A better Hash Method for high-fidelity Monte Carlo simulations on nuclear reactors, Front. Energy Res. **11**, 1861 (2023)
33. D. She, K. Wang, J. Sun, Y. Qiu, Improved methods of handling massive tallies in reactor Monte Carlo Code RMC, in *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, (American Nuclear Society (ANS), Idaho, USA, 2013)
34. M.A. Smith, E.E. Lewis, Na B-C, Benchmark on Deterministic Transport Calculations without Spatial Homogenisation: A 2-D/3-D MOX Fuel Assembly Benchmark, Tech. Rep. NEA/NSC/DOC(2003)16, 2003
35. E. Woodcock, T. Murphy, P. Hemmings, S. Longworth, Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry, in *Proceeding of the conference on Applications of Computing Methods to Reactor Problems,* (Argonne National Laboratory, USA, 1965)
36. Q. Guo, Z. Chen, Multi-regional delta-tracking method for neutron transport tracking in Monte Carlo criticality calculation, Sustainability **10**, 2272 (2018)
37. D. Lee, K. Smith, J. Rhodes, The impact of ^{238}U resonance elastic scattering approximations on thermal reactor doppler reactivity, Ann. Nucl. Energy **36**, 274 (2009)
38. P.K. Romano, J.A. Walsh, An improved target velocity sampling algorithm for free gas elastic scattering, Ann. Nucl. Energy **114**, 318 (2018)
39. W. Rothenstein, Neutron scattering Kernels in pronounced resonances for stochastic doppler effect calculations, Ann. Nucl. Energy **23**, 441 (1996)
40. F.B. Brown, New Hash-Based Energy Lookup Algorithm for Monte Carlo Codes. Los Alamos National Laboratory, Tech. Rep. LA-UR-14-24530, 2014
41. N. Choi, H.G. Joo, Relative speed tabulation method for efficient treatment of resonance scattering in GPU-based Monte Carlo neutron transport calculation, Nucl. Sci. Eng. **195**, 954 (2021)
42. A. Zoia, E. Brun, C. Jouanne, F. Malvagi, Doppler broadening of neutron elastic scattering kernel in Tripoli-4®, Ann. Nucl. Energy **54**, 218 (2013)
43. T. Mori, Y. Nagaya, Comparison of resonance elastic scattering models newly implemented in MVP continuous-energy Monte Carlo code, Nucl. Sci. Technol. **46**, 793 (2009)
44. M.L. Menéndez, J.A. Pardo, L. Pardo, M.C. Pardo, The Jensen-Shannon divergence, Franklin Inst. **334**, 307, (1997)

Cite this article as: Muhammad Rizwan Ali, Murat Serdar Aygul, Deokjung Lee. GREAPMC: development status, capabilities, and future developments, EPJ Nuclear Sci. Technol. **11**, 56 (2025). https://doi.org/10.1051/epjn/2025046