RESEARCH ARTICLE

Development and verification of Hornet – A CUDA-optimized, Monte Carlo-agnostic, continuous-energy cross-section processing code

[version 1; peer review: 2 approved with reservations]

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

[1]Department of Nuclear Engineering, Ulsan National Institute of Science and Technology, Ulsan, 44919, South Korea
[2]Advanced Nuclear Technology and Services, Jung-gu, Ulsan, 44429, South Korea

v1 **First published:** 28 Jul 2025, **3**:34
https://doi.org/10.12688/nuclscitechnolopenres.17676.1
Latest published: 28 Jul 2025, **3**:34
https://doi.org/10.12688/nuclscitechnolopenres.17676.1

Abstract

Background

Hornet is introduced as a GPU-optimized nuclear data processing code designed to accelerate neutron cross-section handling and A Compact ENDF (ACE) file sampling for incident neutron data and thermal scattering laws. Hornet supports continuous-energy Monte Carlo applications and is Monte Carlo–agnostic, making it versatile for academic use and easy integration into various Monte Carlo frameworks.

Methods

Hornet is implemented in CUDA C++ using an object-oriented design to take full advantage of Graphical Processing Unit (GPU) parallelism. Verification was conducted through both standalone tests and integration with GPU-optimized REActor Physics Monte Carlo (GREAPMC) – an in-house Graphical Processing Unit (GPU) Monte Carlo code – across standard benchmark simulations, including International Atomic Energy Agency (IAEA) INDC (USA)-107 pin-cell and Mosteller benchmarks. Performance and precision were assessed by comparing Hornet's output against that of Monte Carlo Simulation (MCS), an in-house CPU-based high-fidelity Monte Carlo code.

Results

Across all benchmark scenarios, Hornet matched MCS in accuracy,

Open Peer Review

Approval Status ? ?

	1	2
version 1 28 Jul 2025	? view	? view

1. **Ho Jin Park**, Kyunghee University College of Engineering, Yongin-si, South Korea
2. **Ilham Variansyah**, Oregon State University, Corvallis, USA

Any reports and responses or comments on the article can be found at the end of the article.

delivering Root Mean Square (RMS) power differences below 0.1% in pin-by-pin power distribution within an OPR1000 fuel assembly. Remarkable tracking rate enhancements were also observed compared to MCS. Hornet's ability to maintain continuous-energy treatment on GPU hardware was demonstrated with precise modeling of bound and free thermal scattering effects.

Conclusions

Hornet enables robust and efficient GPU-based continuous-energy nuclear data processing, producing high-precision results on par with CPU-based Monte Carlo codes. Its object-oriented CUDA C++ architecture ensures maintainability and adaptability. Tested extensively with GREAPMC, Hornet proves to be a powerful tool for modern nuclear analysis and reactor design. Planned enhancements such as support for photo-atomic ACE file processing will broaden its applicability further.

Keywords
Monte Carlo, Particle transport, HORNET, High-performance computing, GPU.

Corresponding author: Deokjung Lee (deokjung@unist.ac.kr)

Author roles: Ali MR: Software; **Aygul MS**: Software; **Lee D**: Software, Supervision

Competing interests: No competing interests were disclosed.

Grant information: This work was supported by an Innovative Small Modular Reactor Development Agency grant (RS-2023-00265742) funded by the Korean government.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Ali MR, Aygul MS and Lee D. **Development and verification of Hornet – A CUDA-optimized, Monte Carlo-agnostic, continuous-energy cross-section processing code [version 1; peer review: 2 approved with reservations]** Nuclear Science and Technology Open Research 2025, **3**:34 https://doi.org/10.12688/nuclscitechnolopenres.17676.1

First published: 28 Jul 2025, **3**:34 https://doi.org/10.12688/nuclscitechnolopenres.17676.1

1. Introduction

During the random walk in a nuclear reactor core, the particle energy, direction, and position change. The neutron transport equation governs the behavior of neutrons in the nuclear reactor core. The numerical solutions of the transport equation comprise deterministic methods. They usually require discretization of a particle's energy, position, and direction. In addition, some of these methods simplify the problem geometry. All of these approximations aim to reduce the complexity of the problem, memory utilization, and computational burden. In contrast, the Monte Carlo (MC) method treats complex geometries with minimal assumptions. Moreover, the continuous treatment of energy, direction, and position reduces the errors introduced by discretization. Unlike deterministic methods, the MC method directly simulates the actual particle behavior. However, the results are prone to statistical errors because of the stochastic nature of the method. A large number of particles must be simulated to reduce the statistical error and achieve accurate results. Despite this, the MC method is widely used across various scientific fields because it is general-purpose and highly accurate, making minimal assumptions about the particle behavior. It is particularly well-suited for the computationally intensive task of simulating the random walk of neutrons in a nuclear reactor core.

Continuous energy cross-sections are fundamental to the Monte Carlo (MC) method in reactor physics. Microscopic cross sections are the interaction probability of a specific isotope with a neutron. A large cross-section indicates that the probability of interaction is large. There are many types of interactions (reactions) between isotopes and neutrons, such as fission, scattering, and absorption. An isotope-specific cross-section lookup is required to obtain microscopic cross-sections at a given energy. These cross-sections are stored in tabular form at discrete energy points, often numbering in thousands, particularly for isotopes with resonance regions where the cross-section changes rapidly. In such cases, finer energy bins were created. These energy bins or points constitute the *energy grid.* Each isotope has a unique temperature-dependent energy grid, and the number of energy points typically varies among the isotopes. The cross-section lookup operation involves locating the particle energy within this grid, followed by linear interpolation between adjacent cross-section values to obtain the cross-section at the energy of interest, as shown in Figure 1. The required microscopic cross-section was calculated using Equation (1).

$$\sigma_n = \frac{E_n - E_i}{E_{i+1} - E_i}(\sigma_{i+1} - \sigma_i) + \sigma_i, \tag{1}$$

where σ_n is the cross-section at energy E_n. This process is repeated for each of the reaction types that are important to the problem (e.g., total, fission, elastic, and absorption).

MC codes rely on macroscopic cross-sections to simulate particle interactions within a medium during random walking. These interactions encompass particle absorption, elastic and inelastic scattering, fission reactions, and escape from a medium. The macroscopic cross-section depends on both the properties of the medium, such as the composition and temperature, and the properties of the particles, such as the energy. The macroscopic cross sections of a material are calculated as the sum of the products of the microscopic cross sections and number densities of the constituent isotopes, as shown in Equation (2).

$$\sum_x = \sum_{k=1}^{n\_isotopes} N_k \sigma_x, \tag{2}$$

where the material is assumed to have *n_isotopes* number of total isotopes as a constituent of the material, and x is the reaction type. For instance, if UO_2 fuel has five isotopes (^{234}U, ^{235}U, ^{236}U, ^{238}U, ^{16}O), the total macroscopic cross-section is calculated by adding the product of the individual total microscopic cross-section and number density.

Most of the runtime in the Monte Carlo (MC) code is spent on continuous-energy cross-sectional lookups.[1] In addition to the cross-section calculation, a continuous-energy cross-section treatment code samples the energy and angular distributions for reactions that produce secondary neutrons, such as fission, inelastic, and (n,xn) reactions. Furthermore, the code must handle the analytical calculation of the outgoing energy in the case of elastic scattering reactions.

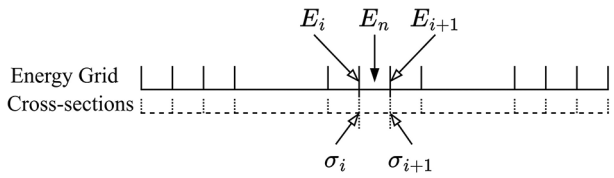

Figure 1. Linear interpolation is employed to calculate cross-section at the desired energy.

CPU-based computer clusters are often utilized to reduce the simulation time of the Monte Carlo (MC) method while preserving the accuracy. However, these clusters have a scalability limit; beyond a certain number of nodes, the communication overhead between nodes can negatively impact the performance. By contrast, graphical processing units (GPUs) with thousands of cores offer an excellent alternative to traditional CPU systems.

The computer science community has been using GPUs for scientific computations. More recently, the reactor physics community has begun to explore the benefits of GPUs, achieving significant speedups in various applications.[2,3] While research on GPU-accelerated Monte Carlo (MC) methods often emphasizes results, less attention has been paid to implementation details. To the best of our knowledge, prior reactor physics research has not specifically focused on the independent development of GPU-optimized continuous energy cross-section codes; instead, discussions typically occur within the broader context of the MC process. The CORE lab at UNIST addressed this by developing Hornet, a GPU-accelerated continuous energy cross-section code designed to support comprehensive GPU-based MC methodologies. This study focuses on the development and verification of Hornet, which can operate as a standalone code (for academic purposes) and provides an interface for integration with GPU-enabled MC codes. Hornets are fully object-oriented and are written in CUDA C++. Several programming models, including Kokkos,[4] OCCA,[5] RAJA,[6] OpenMP,[7] OpenACC,[8] SYCL,[9] and OpenCL[10] enable the use of NVIDIA GPUs. However, because CUDA was developed and supported by NVIDIA, it offers a level of control over implementation details and GPU support unmatched by these alternatives. Therefore, CUDA was selected as the platform for development. Furthermore, given that the code in this study was written from the ground up, a CPU-first and then a GPU-accelerated approach using directives was deemed inefficient. Finally, the CUDA API provides access to numerous optimized libraries such as cuFFT,[11] cuRAND,[12] cuSPARSE,[13] and cuBLAS,[14] which can be employed when required.

The rest of this paper is organized as follows. Section 2 reviews the existing literature on GPU-enabled continuous energy treatment. Section 3 details the Hornet implementation. Section 4 presents a thorough verification of the Hornet. Finally, Section 5 concludes the paper with a summary of the key findings and potential future research directions.

2. Background

The MC code ubiquitously fetches continuous energy cross-section data from (A Compact ENDF) file format.[15] ACE data originate from the data given in the ENDF-6 format.[16] However, the evaluated nuclear data Files) data cannot be directly used in the MC code. Hence, NJOY[17] or FRENDY[18] was employed to process the data and convert it into the ACE format.

Open-source nuclear data libraries (NDLs) exist, such as ACEtk[19] and Papillon,[20] or are integrated into open-source Monte Carlo codes such as OpenMC[21] and SCONE,[22] which are often written in FORTRAN or rely heavily on programming features incompatible with CUDA. More importantly, they do not leverage support for CUDA and are not designed with GPU-optimized strategies. The following is a concise overview of the key components of an NDL.

2.1 Energy grid lookup

The pros and cons of different methods devised to handle look-ups in the context of GPU are briefly discussed here.

2.1.1 Binary search

A fundamental approach for locating bounding energy bins is a binary search[23] of the energy grid. This method has a time complexity of $O(\log n)$, where n represents the size of the energy-grid array.[23] Binary search is advantageous owing to its simplicity and minimal memory overheads. However, performing binary searches on large energy grid arrays negatively impacts GPU performance owing to strided memory accesses. Strided accesses, where data are not stored contiguously in memory, can significantly increase the read times.

Figure 2 illustrates the difference between coalesced and strided memory accesses. The resulting poor cache utilization and limited global memory throughput make binary searches an inefficient choice for GPUs.

2.1.2 Unionized grid method

The unionized grid method[24] was first introduced in Serpent MC code.[25] As the name suggests, this method creates a union of the energy grid points of all nuclides in the problem. The individual cross-sections for all nuclides were then stored over this unionized energy grid. A single binary search to determine the bounding energies is sufficient for all isotopes.[24] This method has the major drawback of large memory requirements. Because this method stores a value for

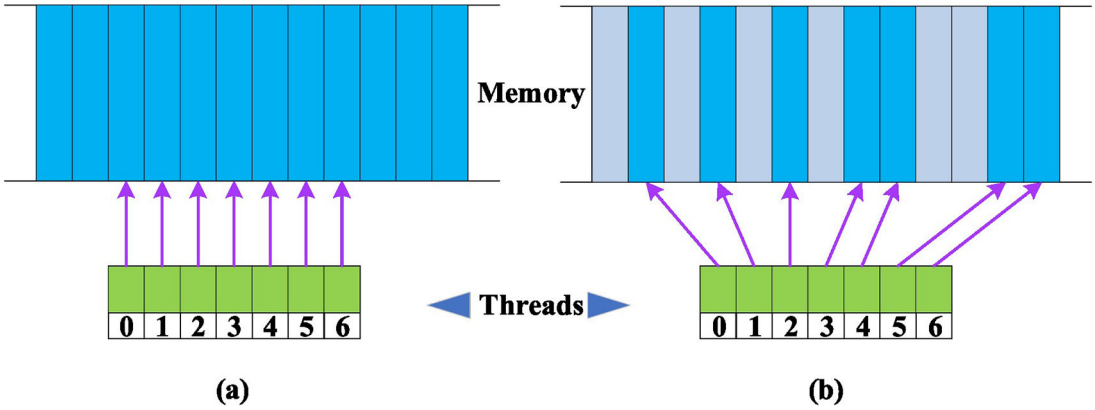

Figure 2. Example of memory access (a) coalesced and (b) strided.

each energy grid point, the matrix of the cross-sections will mostly be sparse. This situation is further aggravated when the problem demands the storage of cross sections at multiple temperatures.[26]

2.1.3 Double index scheme

An improvement over the unionized energy-grid method resulted in the development of a double-indexing method.[24] Instead of storing the cross-sections directly on the unionized grid, the pointers to the individual energy grids of the nuclides are stored on the unionized grid in the form of a double index table. Once the unionized grid point is located via a binary search, the local grid point of each nuclide is determined by examining the corresponding entry of the double index table. The double-index table reduces the memory usage of the unionized grid method; however, the size of the table is the product of points in the unionized grid and the number of problem nuclides. This will particularly explode depletion problems.

2.1.4 Fractional cascading

To reduce the memory requirements while achieving the computational performance of the unionized grid method, Lund et al.[27] proposed a cascade grid-based energy lookup. This method increases the memory by only a factor of two compared to the individual energy grids. The cascade grid consisted of the original nuclide energy grids, an additional set of energy grids (called augmented grids), one for each original energy grid, and a couple of pointers for each entry in the augmented grids. The augmented grids are created by constructing a unified grid for the first and second isotopes, followed by another grid for the second and third isotopes. The first pointer stored the energy index in the original nuclide grid, whereas the second pointer provided the energy index in the next augmented grid. The performance indicates a speedup of approximately 1.2 to 1.5 against using individual nuclide grids alone.[27] The fractional cascading technique significantly reduces memory usage compared to the global unionized method. However, this method lacks memory coalescence, as the second pointer always points to the next augmented grid.

2.1.5 Hash-based energy lookup

A hash function is an algorithm that takes the input and produces a hash value. This value uniquely represents the input data.[28] The purpose of a hash function is to efficiently map large amounts of data to smaller fixed-size values, making it easier to store, retrieve, and compare data.

This method employs a hash-based approach to expedite the energy grid search by mapping neutron energy to the indices of the hash grid.[29,30] The hash grid indices were then mapped to the energy points on the nuclide energy grid. By narrowing down the search space to a subset of energies, the algorithm can efficiently locate relevant cross-sectional data.[30] This logarithmic mapping technique divides the energy range into equal lethargy intervals. Walsh et al.[1] outlined the algorithm well, and the steps were as follows for a single nuclide.

1. First, the equal-logarithmic energy spacing was determined using Equation (3). The minimum and maximum energies are predetermined for the problem and supplied to the function that builds the hash function.

$$\Delta u = \frac{\log\left(\frac{E_{\max}}{E_{\min}}\right)}{N_{Bins}}. \tag{3}$$

2. Next, the indices of an individual nuclide's energy grid are determined using Equation (4) and stored as an array (hash table).

$$E^{H}_{N_{bins}-i+1} = \frac{E_{\max}}{\exp(i\Delta u)}. \text{ where i} = N_{bins}, N_{bins}-1, \ldots, 1, 0 \tag{4}$$

3. During the simulation, suppose the neutron energy at which the cross-sections are required is E_n. Then, the indices in the hash table are determined directly using Equation (5).

$$k(E_n) = N_{bins} - \left\lfloor \frac{\log\left(\frac{E_{\max}}{E_n}\right)}{\Delta u} \right\rfloor. \tag{5}$$

4. Now, the binary search is performed on $[E(k(E_n))\ E(k(E_n)+1)]$.

The original method[30] builds a hash table on equi-lethargy intervals. Wang et al. compared the energy lookup methods for CPU and Many Integrated Core (MIC) architectures and proposed a new method, namely the N-ary map for Single Instruction Multiple Data (SIMD) architectures, where the hash table uses an equal number of energy intervals.[31] Chen et al. proposed two optimization strategies for existing energy lookup algorithms [https://www.sciencedirect.com/science/article/abs/pii/S001046551830273X]. These are the neighboring material cascade grid (NMCG) and Adaptive Optimal Logarithmic Grid (AOLG). AOLG can be useful for SIMD architecture, as for some isotopes, the resonance region can have hash bins with thousands of energy points. Consequently, an unbalanced distribution of the energy bins can cause thread divergence.

Raffuzzi et al. proposed accelerating the MC code by approximating the thermal cross-sections with functional forms.[32] In the thermal region, the cross-section variation is smooth; hence, the thermal region can be represented using functions. However, because this method directly replaces cross-sectional data with an approximation, the method can affect the accuracy of the results.[32]

2.2 GPU-specific continuous-energy treatments

This subsection discusses continuous-energy cross-section treatments in state-of-the-art Monte Carlo codes. From a literature point of view, none of these codes uses a Monte Carlo agnostic approach to the continuous-energy cross-section code.

2.2.1 WARP

WARP is among the pioneering continuous-energy codes specifically designed for GPUs.[33] WARP uses PyNE ((Python for Nuclear Engineering)[34] to read all cross-section data and optimize data storage by unionizing the energy grids of the problem nuclides and employing CUDA's *float4* to store microscopic cross-sections. The float4 construct maximizes bandwidth as more data are loaded in one memory transaction. Additionally, in WARP, the total cross-sections of all the isotopes are stored together within the unionized energy grid to increase coalescence. The WARP also devised a storage strategy to tabulate the outgoing angle and incident neutron energy for secondary angle-energy distributions. This strategy creates two additional matrices, each with dimensions matching the unionized cross-section matrix. These matrices contain pointers for the corresponding distribution data of a specific reaction. Two distinct matrices are necessary: one for scattering distributions, and the other for energy distributions.

2.2.2 Shift

Shift was the first continuous-energy Monte Carlo code optimized for running on both CPU and GPU architectures. Both versions used the equi-lethargy hash-based energy look-up method with 16,384 bins.[35] The benefit of a hash-based

energy look-up is a smaller memory footprint compared to the other methods. Although the requirement to search for individual hash functions can reduce performance, the reduced search space can compensate for this performance loss.

2.2.3 PRAGMA

PRAGMA is a GPU-specific MC code developed and maintained by Seoul National University.[36] Similar to WARP, PRAGMA utilizes the *float4* construct to store primary microscopic cross sections and applies only inelastic scattering to the $S(\alpha,\beta)$ treatment.[37] Additionally, the literature does not mention any treatment for the unresolved resonance region in PRAGMA. For energy lookup, PRAGMA employs a linear-interval hashing scheme.[36]

2.2.3.1 Linear Interval Hashing Scheme

This method is a spin-off of a double-index scheme. This variant of the double-index scheme increases memory coalescence and reduces the memory burden of the conventional scheme. In this method, every N^{th} value from the unionized grid was stored. The pointers corresponding to the individual energy grids do not point to each energy value. Therefore, a linear search was adopted for the local energy grid after a lower grid point was identified from the unionized grid. The value N is called a hash, and the purpose of using hashed storage is to reduce the memory requirement. Additionally, fewer points in the unionized grid could potentially increase the memory coalescence.

2.2.4 OpenMC

Two versions of GPU-enabled OpenMC exist: one leverages OpenMP[7] to offload the code to the GPU,[38,39] whereas the other uses CUDA C++.[40] OpenMP eliminates polymorphisms in classes within the nuclear data hierarchy. By contrast, the CUDA version maintains polymorphism by creating objects directly on the device and implementing a polymorphism-supporting C++ unique pointer. In addition, the CUDA version uses single-precision nuclear data rather than double-precision data.[40]

2.2.5 Mercury

Lawrence Livermore National Laboratories (LLNL) is actively working on porting their codes Mercury and Imp to GPUs.[41–43] Mercury uses GIDI,[44] a nuclear data library developed and maintained by LLNL, and uses the Generalized Nuclear Database Structure (GNDS) format.[45] GIDI uses virtual functions; however, virtual function objects do not copy correctly from the CPU to the GPU using the managed memory. Hence, serialized functions are introduced along with overloading the new operator to copy data to the GPU.[41]

3. Methods and features of Hornet

Hornet was developed with two main objectives: the primary objective was to support the transition of UNIST's in-house GPU-optimized REActor Physics Monte Carlo (GREAPMC) code from multigroup to continuous-energy treatment.[2] The secondary objective was to use the code for academic purposes, allowing students to engage in continuous-energy neutron physics using GPUs.

3.1 Salient features

3.1.1 Energy look-up

We implemented four energy look-up schemes in Hornet: unionized grid, double index scheme, equi-lethargy hash-based scheme, and linear interval hashing scheme. Because the purpose of this work is verification, the equi-lethargy hash-based algorithm devised by Brown[30] was used as the default option. The hash function is built on an individual energy grid. As recommended by Brown,[30] we employed 8000 equal-lethargy intervals. All objects are formed in the host and then transferred to the device; hence, the hash function is built in the constructor of the top-level class.

3.1.2 Thermal scattering

The impact of the chemical binding of the target molecules is strong for thermal neutrons, especially for energies below 4 eV. Because such scattering kinematics are more profound in moderating materials, it is important to account for these effects to obtain accurate results. For a thermal neutron, the de Broglie wavelength of the neutron and the interatomic spacing of the material coincide.[46] This leads to neutron interactions with the lattice structure of the material, in addition to the target nuclei. The thermal scattering laws (TSL) in the ENDF considers these additional effects and tabulates the cross

sections, outgoing energy distribution, and angular distributions for thermal scattering with such materials. While the theory and derivation of the TSL are out of scope, the reader is referred to the articles by Tang et al.[47] and Fleming et al.[48] or, for a very detailed theory and derivations, to the book by Squires.[49] The S(α, β) table contains three types of thermal-scattering reactions.[50]

1. Inelastic scattering
2. Coherent elastic scattering
3. Incoherent elastic scattering

In addition to the elastic and inelastic cross sections, S(α, β) also contains information about the energy-angle distributions. It is important to distinguish between the elastic/inelastic terminology used in this study and the conventional elastic/inelastic concept. The elastic and inelastic terms in the context of S(α, β) refer to the excitation of the scattering system, whereas conventionally they refer to the excitation of the target nuclei. Coherent elastic scattering occurs in crystalline structures, incoherent elastic scattering reactions occur in hydrogen-rich solids, and inelastic scattering is important for other materials, including hydrogen in water. Although only inelastic scattering treatment is important for LWRs, all three scattering reactions are implemented in Hornet for future applications in other reactors.

3.1.3 Unresolved resonance treatment

The energy range of the cross-sections is divided into distinct regions: thermal, resolved resonance, unresolved resonance, and fast (continuum) regions. The resonance peaks in the unresolved resonance region (URR) are closely spaced such that they become indistinguishable. Probability tables (PTs), which represent cross sections using average resonance parameters, were employed to address this issue. The primary criterion for PTs is the preservation of energy self-shielding.[51] Nuclear data processing codes, such as NJOY, generate data for URR using unresolved resonance parameters. These data are used to create cumulative distribution functions (CDFs), which are then translated into tabular cross-sections dependent on the incident energy.[52] This method is almost ubiquitous for the probabilistic representation of the URR region in MC codes, and the same is adopted in Hornet. Details can be found in.[53,54] This section provides a brief overview of the study.

Starting with the ACE file, the range of the cross-sectional values is divided into bands. Each band contained cross-sectional values or factors for a single probability. A probability table was created for each incoming energy source. The probability table holds the CDFs and bands. Hence, this is a three-tier hierarchy, where the lowermost level represents the bands. The class for the probability table forms the medium level, and the top level is where the incoming energy and corresponding table are stored. Figure 3 illustrates a unified modeling language (UML) class diagram for URR in Hornet.

3.1.4 Secondary angle distributions

The fission, inelastic, and (n,xn) reactions involve secondary neutrons. Hence, it was necessary to sample the secondary angles and energy distributions. An ACE file determines whether the angle and energy must be specified separately or as a correlated angle-energy distribution. In cases in which they are uncorrelated, the secondary angle distribution is sampled independently. Similarly, the outgoing energy is calculated analytically for elastic scattering; therefore, only the secondary angle distribution must be specified. In these scenarios, the angle distribution is represented in one of the following formats:

1. Isotropic distribution
2. Equiprobable distribution
3. Tabular distribution

3.1.5 Outgoing energy and angle

Whenever such a reaction generates secondary neutrons, it is mandatory to determine the outgoing neutron energy and angle. In addition to elastic scattering, the outgoing neutron energy for other reactions is determined in one of the following ways. These methods are known as ACE laws[15] and are based on ENDF format.[16] For a specific isotope,

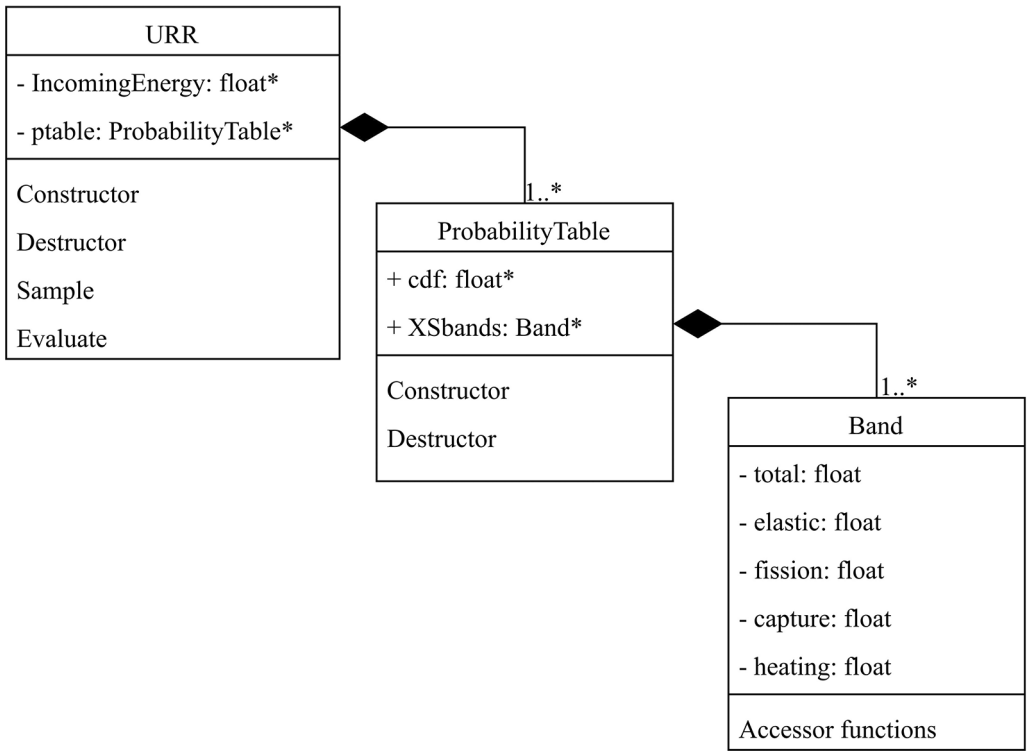

Figure 3. UML class diagram for URR treatment in Hornet.

information regarding the selection of the secondary energy distribution is provided in the ACE file. The ACE laws implemented in Hornet are as follows.

1. Equiprobable Energy Bins (ACE Law 1)
2. Inelastic Level Scattering (ACE Law 3)
3. Continuous Tabular (ACE Law 4)
4. Maxwell Spectrum (ACE Law 7)
5. Evaporation Spectrum (ACE Law 9)
6. Energy-dependent Watt Spectrum (ACE Law 11)
7. Kalbach-Mann law (ACE Law 44)
8. Correlated Energy and Angle (ACE Law 61)
9. N-Body Phase Space (ACE Law 66)

3.2 Implementation details

Hornet was developed using an object-oriented programming (OOP) approach to simplify maintenance and debugging. The inherent modularity of the continuous-energy cross-section processing lents itself well to the OOP. For example, each secondary distribution is implemented as a distinct class based on its function. These classes are linked through associations, thereby creating a hierarchical dependency structure. Consequently, adapting OOP-based code for GPU execution presents a significant challenge.

3.2.1 Memory management

The primary challenges in porting Hornet to GPU are memory allocation, data transfer, and deallocation. These three CUDA functions form the basis of memory management in CUDA C++.

1. `cudaMalloc()`: Allocates the device memory and assigns the address of the data to the device pointer.
2. `cudaMemcpy()`: Copies the data between host and device. This study makes four arguments.
   a. Destination memory address
   b. Host memory address
   c. Size in bytes to copy
   d. Type of transfer
3. `cudaFree()`: Frees the memory space indicated by the device pointer. This device pointer must have been returned by call to `cudaMalloc()`.

If all data transfers are centralized in one location, such as in `main()`, the device pointer is consistently available, making memory transfer and deallocation operations straightforward. However, when there is a tree-like dependency structure between classes, managing the device memory becomes cumbersome. Hornets feature a hierarchical class structure, with each isotope having numerous root-level objects that must be handled. Therefore, Hornet employs a special memory management scheme in which each class is equipped with two functions: `transfer_to_device()` and `free_device_memory()`. As the names suggest, these functions are responsible for transferring data to the device and freeing its memory. Breaking down the working logic into steps, starting from a top-tier class to a lower-most class for a mock-up example, will help grasp the memory management in Hornet.

3.2.1.1 Transferring data to device

The `transfer_to_device()` function accepts an argument, an object of the same class, passed by the reference. This object is used to allocate the device memory, and the data from the host object (that invoked the function) are copied to the object passed as an argument. Suppose there are two classes. The `main` function() is at the top of the hierarchy, as shown in Figure 4. From within the `main()`, an object of the `MaxwellEnergy` class `h_ME` is instantiated. The object exists on the host side. A pointer `d_ME`, which acts as a device pointer, is also created. The data to which the device pointer will point are carried in the object `dh_ME`. The function responsible for allocating the device data and transferring the host object's data to the members of `dh_ME` is invoked by object `h_ME`, and object `dh_ME` is passed as an argument. Subsequently, the program launches a kernel, performs some processing, and frees the memory using the object `d_ME`.

```
int main(void)
{
    MaxwellEnergy h_ME(/* Pass ACE File */);
    MaxwellEnergy* d_ME = nullptr;
    cudaMalloc(&d_ME, sizeof(MaxwellEnergy));
    MaxwellEnergy dh_ME;
    // Transfer data from h_ME to dh_ME
    h_ME.transfer_to_device(dh_ME);
    // Assign an address to d_ME
    cudaMemcpy(d_ME, &dh_ME, sizeof(MaxwellEnergy), cudaMemcpyHostToDevice);
    /*** Launch CUDA Kernel ***/
    // Free the device memory
    dh_ME.free_device_memory();
    cudaFree(d_ME);
    return 0;
}
```

Figure 4. The `main()` function of the example.

```
class MaxwellEnergy
{
    private:
        Tabulated* theta {nullptr};

    public:
        MaxwellEnergy() = default;
        MaxwellEnergy(/*Takes ACE file as an argument and allocates theta*/);
        void transfer_to_device(MaxwellEnergy& data);
        void free_device_memory();
};
```

Figure 5. Header file of the `MaxwellEnergy` class.

```
void MaxwellEnergy::transfer_to_device(MaxwellEnergy& data)
{
    Tabulated dh1;
    cudaMalloc(&data.theta, sizeof(Tabulated));
    theta -> transfer_to_device(dh1);
    cudaMemcpy(data.theta, &dh1, sizeof(Tabulated), cudaMemcpyHostToDevice);
}
void MaxwellEnergy::free_device_memory()
{
    Tabulated t1;
    cudaMemcpy(&t1, theta, sizeof(Tabulated), cudaMemcpyDeviceToHost);
    t1.free_device_memory();
    cudaFree(theta);
}
```

Figure 6. Implementation of the `MaxwellEnergy` class.

The `MaxwellEnergy` class forms the 2nd level of the hierarchy. Figure 5 shows the header file of the `MaxwellEnergy` class. This class has only one data member, which is a pointer of the `tabulated` class. The object of the `tabulated` class was dynamically created within the constructor of the `MaxwellEnergy` class.

Figure 6 illustrates the implementation of `MaxwellEnergy` class functions. Within the scope of `transfer_to_device()`, an object of the `tabulated` class is created (`dh1`). The `transfer_to_device()` function of the `tabulated` class was invoked using the host-side tabulated object (`theta`). Object `dh1` contains the device data of the `tabulated` class. This device memory was assigned to `data.theta` using `cudaMemcpy()`. After the scope of the function ends, the `dh1` object is destroyed, but the device data are stored in `data.theta`.

3.2.1.2 Freeing the device memory

The deallocation of device memory is impossible without a pointer that stores the address of the device data. To circumvent the issue arising when the device pointer needed to free the memory is unavailable, we used `cudaMemcpy()` to copy the device pointer to the host memory. It is pertinent to mention that `cudaFree()` always requires a pointer to reside on the host. In Figure 6, `theta` is currently residing on the device; therefore, it cannot be directly used to free the `tabulated` object's device memory. To free the `tabulated` object inside the `MaxwellEnergy` object, we first copy the object from the device to the host using `cudaMemcpy()`, as shown in the `free_device_memory()` function in Figure 6. The `free_device_memory()` of the `tabulated` class is then invoked using the host object `t1`. The `Tabulated:free_device_memory()`, in turn, uses `cudaFree()` on the `tabulated` data members.

Thus, the `main()` function remains clutter-free, and all data management is transparent to the `main()`. In the current framework, GREAPMC, through an interface class, invokes only the `transfer_to_device()` function of the top-level class, and the remaining object creation, memory allocations, and data transfers are automatically completed. Similarly, to free the device memory, GREAPMC simply invokes the `free_device_memory()` for each of the objects created at the start of the simulation and the device memory deallocation, and the destruction of the objects on the device is transparently completed. We have checked the method for memory leakage using compute-sanitizer[55] and no memory leaks or segmentation errors were detected, asserting the correctness of the method.

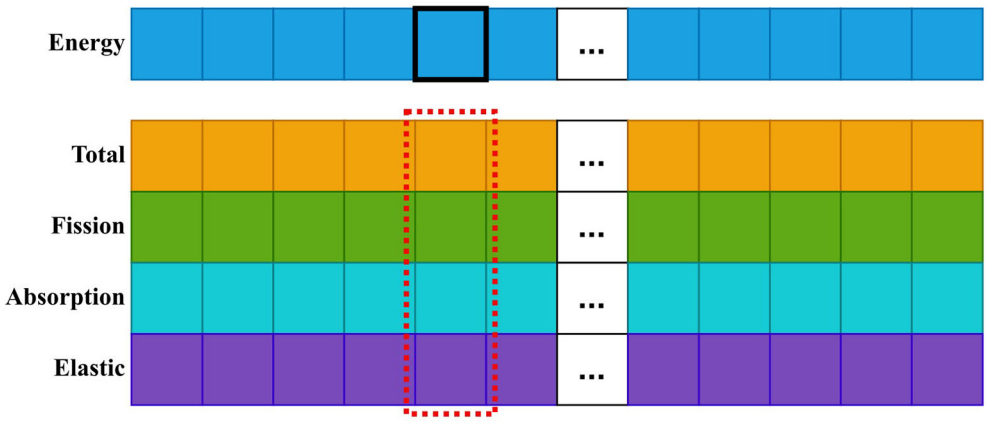

Figure 7. Schematic diagram of `float4` structure.

3.2.2 Optimizations

The general requirements for the memory layout and access patterns of GPUs fundamentally differ from those of CPUs. Hence, in this context, the optimizations ubiquitously used for continuous energy cross-section treatment are given here.

3.2.2.1 Primary cross-sections

Similar to WARP[33] and PRAGMA,[36] the four primary microscopic cross-sections (total, elastic, absorption, and fission) are stored together in the *float4* structure for each isotope in the problem.

Float4 is a structure that stores four floating-point (float) values packed together, enabling efficient memory access and vectorized operations on GPUs. This data structure is optimized for GPU architectures, allowing a single instruction, such as an *LD.128*-bit read, to load all four values into the memory simultaneously.

The size of the *float4* arrays corresponds to the length of the energy grid, with each structure storing four values per energy-grid index. Once the energy grid index is determined, a single read simultaneously retrieves all four values. Normally, four values will incur four memory reads, but CUDA issues one load instruction when the data type is *float4* to retrieve four values, as shown in Figure 7. This is a crucial optimization because cross sections are frequently calculated within the transport loop. Given that global memory reads can become a bottleneck, special attention is paid to optimizing this access pattern.

3.2.2.2 Single precision arithmetic

The uncertainties in nuclear data exceed the errors resulting from reduced significant figures when single-precision arithmetic is used instead of double precision. Therefore, Hornet uses single-precision across nuclear data, with the exception of calculating the angle distributions in elastic scattering. For the outgoing angle calculations, it was observed that a single precision led to increased errors in the eigenvalue and pin power calculations. Single precision not only enhances cache hit rates but also reduces memory requirements by half.

4. Results and verification

This section presents the numerical results, verification, and performance evaluation of Hornet in comparison with MCS. To generate verification results, Hornet was coupled with GREAPMC. These results were obtained using standard history-based neutron tracking and an equi-lethargy-based hash method for energy lookup. The ENDF/B-VII.1 library was used, applying linear interpolation for temperature when the material temperature fell between the two available library temperatures. The verification includes the sampling of selected reactions, S(α, β) treatment, resonance scattering, and pin-by-pin power results for a fuel assembly case.

4.1 Sampling of reactions

The three most important isotopes involved in nuclear fission are ^{238}U, ^{235}U, and ^{239}Pu. We sampled the following two reactions with selective incoming energies and temperatures:

1. MT 91 (z,n_c)
2. MT 18 $(z,fission)$

Reactions were sampled with both MCS and GREAPMC using 1×10^6 samples. We employed Jensen–Shannon divergence (JSD)[56] to assess the output's quantitative closeness. The JSD score is a metric used to quantify the similarity between the two probability distributions. It combines information from the Kullback-Leibler divergence in both directions, providing a single measure of dissimilarity. The formula for Jensen-Shannon divergence is provided in Equation (6).

$$JSD(P\|Q) = \frac{K(P\|M) + K(Q\|M)}{2} \tag{6}$$

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

Figure 8 illustrates the outgoing energy for the inelastic reaction MT 91 with ^{235}U at a temperature of 293.6 K. The energy of the incoming neutrons in all the samples was 2.625 MeV. The plot shows good agreement between MCS and Hornet, with a JSD value of 0.1.

Figure 9 shows a plot of the outgoing energy for ^{238}U MT 91 reaction with the same incoming neutron energy as that in the previous case. The temperature was then changed to 600 K. Good agreement between both codes is evident from the JSD value of 0.084.

Figure 10 shows the outgoing energy when a 2.625 MeV incoming neutron interacts inelastically with ^{239}Pu at 900 K. The JSD value for this case was 0.102. The slight difference in reaction sampling may be due to the difference in precision. Hornet uses single precision throughout, except for angle calculation in elastic scattering, whereas MCS uses double precision throughout.

Figure 11 depicts the outgoing energy for the fission reaction with ^{235}U at 293.6 K when a neutron of energy 0.523 MeV interacts. The JSD value for this reaction was 0.13.

Figure 12 illustrates the outgoing energy when MT 18 was sampled for a neutron with an energy of 1.811 MeV interacting with ^{238}U at 293.6 K. The JSD value for this reaction was 0.13, as in the previous case.

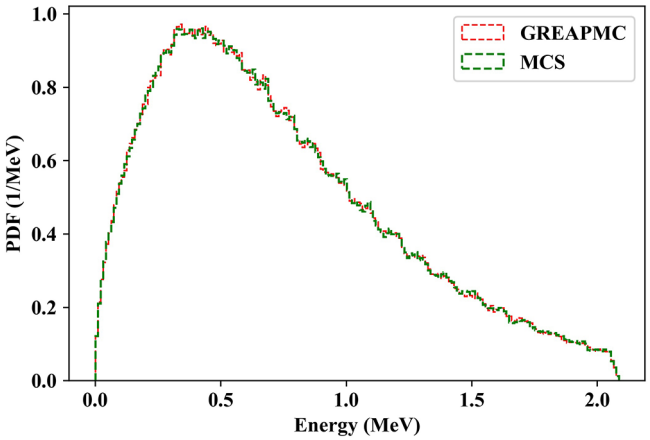

Figure 8. MT-91 reaction with ^{235}U at 293.6 K for an incoming energy of 2.625 MeV.

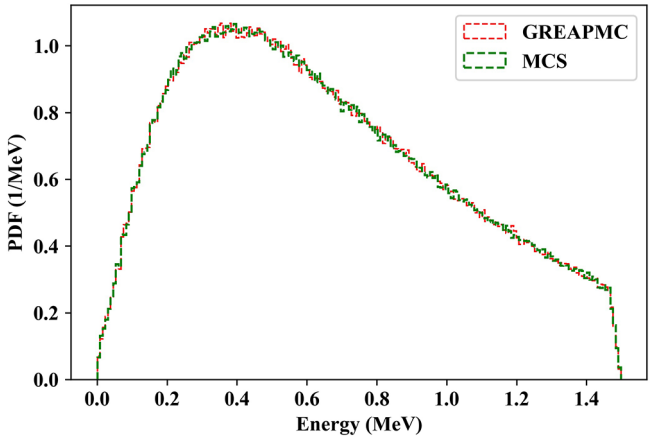

Figure 9. **MT-91 reaction with ^{238}U at 600 K for an incoming energy of 2.625 MeV.**

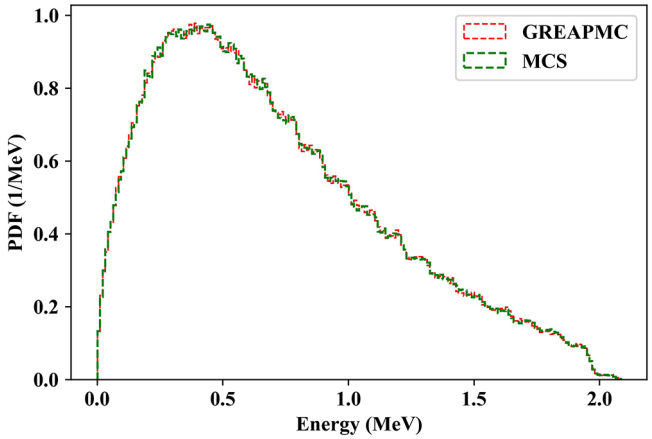

Figure 10. **MT-91 reaction with ^{239}Pu at 900 K for an incoming energy of 2.625 MeV.**

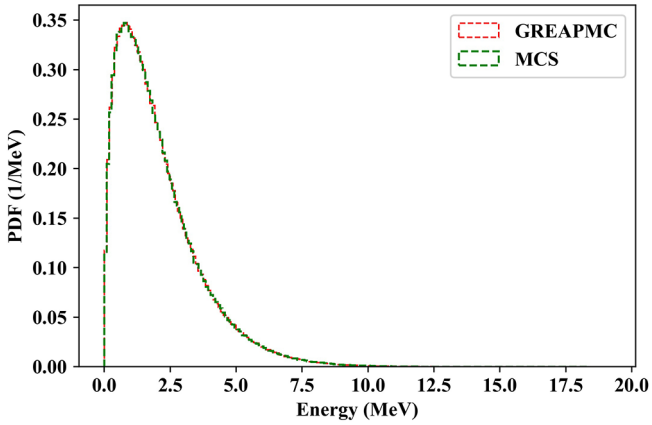

Figure 11. **MT-18 reaction with ^{235}U at 293.6 K for an incoming energy of 0.523 MeV.**

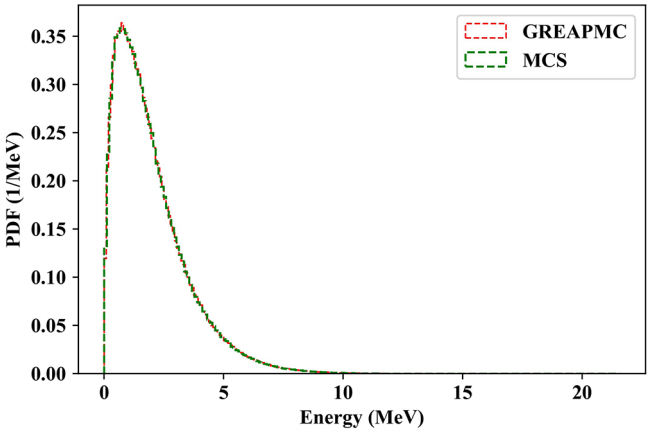

Figure 12. MT-18 reaction with ^{238}U at 293.6 K for an incoming energy of 1.811 MeV.

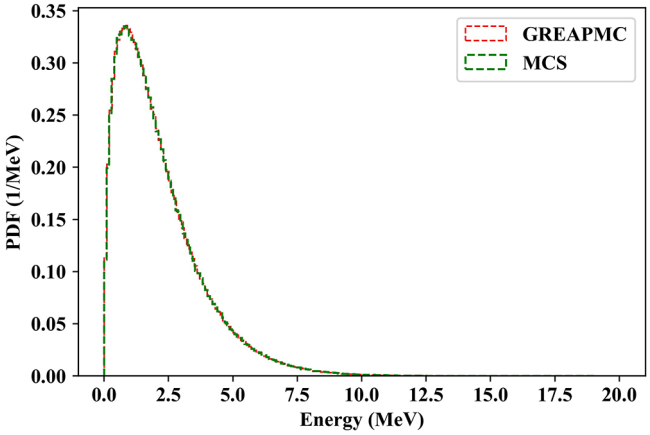

Figure 13. MT-18 reaction with ^{239}Pu at 293.6 K for an incoming energy of 0.505 MeV.

Figure 13 shows the outgoing energy when a neutron with an energy of 0.505 MeV interacts with ^{239}Pu at room temperature. The JSD value for this reaction is 0.13.

Overall, these results indicate a good agreement between Hornet and MCS. The JSD value was approximately 0.1, strengthening the closeness between the distributions.

4.2 Thermal system

To validate the $S(\alpha,\beta)$ treatment and overall eigenvalue, we considered IAEA's International Nuclear Data Committee's INDC (USA)-107 benchmark.[57] This is a simple benchmark that simulates a water-moderated UO_2 pin cell. The square geometry is reflective at all faces and contains only two materials: the fuel and moderator. Each material had two isotopes. The fuel contains ^{235}U and ^{238}U, while the moderator contains ^{1}H and ^{16}O. The benchmark consisted of three problems that differed according to the radius of the fuel rod. Each problem was run with two moderator conditions: free atoms and applying the thermal scattering law. The free-atom condition assumes thermal motion using free atoms, where the atomic motion follows a Maxwellian distribution. The bound state accounts for the atomic translational motion along with vibration and rotation. Figure 14 shows the pin-cell problem with the radius of each problem.

The temperature was isothermal (293.6 K), with a moderator density of 1 g/cc and a fuel density of 18.8 g/cc. The fuel composition is listed in Table 1.

The k_{eff} difference between GREAPMC and MCS is presented in Table 2. The simulations involved one million particles per cycle, with 100 active and 1100 inactive cycles. The k_{eff} difference was computed using Equation (7).

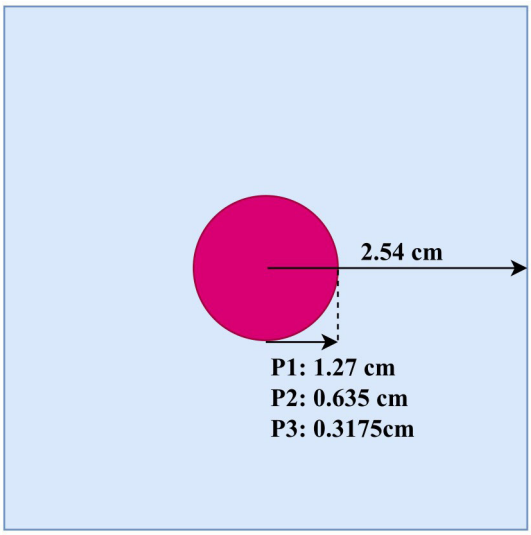

Figure 14. **The cross-sectional view of the pin problem.**

Table 1. **The atomic fractions of Uranium isotopes in each problem.**

Problem	^{238}U (a/o)	^{235}U (a/o)
P1	99.02	0.98
P2	96.5	3.5
P3	30	70

Table 2. **Eigenvalue difference for the three problems.**

Problem	k_{eff} difference (pcm)	
	Bound	**Free**
P1	5.05 ± 2.07	-4.03 ±1.92
P2	1.74 ± 2.68	-0.89 ± 2.58
P3	7.23 ± 3.31	-2.46 ± 3.28

$$k_{eff} \text{ difference [pcm]} = \left(k_{eff}^{S} - k_{eff}^{R}\right) \times 10^{5} \quad (7)$$

Here, S refers to the simulated result (using GREAPMC), and R refers to the reference value (obtained via MCS).

The results show good agreement between GREAPMC and MCS for the bound and free cases. The difference is mainly due to GREAPMC's use of single precision for continuous energy cross-sections, geometry, and material treatment.

4.3 Mosteller Benchmark

The Mosteller benchmark problem[58,59] was proposed to assess the resonance scattering effect of the Doppler broadening rejection correction (DBRC) against the constant cross-section (CXS) approximation at different fuel temperatures.[60] Specifically, the Doppler coefficient was calculated by considering two configurations: hot zero power (HZP) with fuel at 600 K and hot full power (HFP) with fuel at 900 K. All other materials were kept at a temperature of 600 K. The upper energy limits for DBRC were set at 210 eV in MCS and GREAPMC.[61] The benchmark consists of UO_2 fuel, weapons-grade mixed oxide (MOX) fuel, and reactor-recycled MOX fuel. The Doppler coefficient (or fuel temperature coefficient [FTC]) in pcm/K is calculated using Equation (8).

$$\text{Doppler Coefficient} = \frac{10^5}{\Delta T}\left(\frac{1}{k_{HZP}} - \frac{1}{k_{HFP}}\right) \tag{8}$$

The simulations employed three million particles per cycle in 70 inactive and 300 active cycles. Table 3 lists the Doppler coefficients for the UOX fuel. The Uranium enrichment varies from natural uranium to 5%. The maximum relative error between the Doppler coefficients calculated by MCS and GREAPMC is 0.86% for CXS for a 5% enrichment, and 0.65% for DBRC for a 3.9% enrichment case.

Figure 15 illustrates the Doppler coefficients for CXS and DBRC using the MCS and GREAPMC (coupled with Hornet). The Doppler coefficient values for all methods lie within the standard deviation of each code. Additionally, there was no bias in the coefficient values.

The Doppler coefficients for the reactor-recycle MOX fuel are listed in Table 4. The Plutonium enrichment varied from 1% to 8%. The maximum relative difference in the coefficient was 0.56% for CXS (1% enrichment) and 0.5% for DBRC (4% enrichment).

Figure 16 shows the Doppler coefficients for the reactor recycling case. The FTC values from one code lie within the standard deviation of the other codes, indicating excellent agreement. Moreover, there was no bias in the calculation of FTC.

Table 5 gives the FTC for the weapons-grade MOX fuel for different plutonium encirclements. The FTC was the most negative for 2% plutonium enrichment. The maximum relative difference in CXS was 0.31% at 1% plutonium enrichment. Similarly, the maximum relative difference for the DBRC was 0.42% in the 1% enrichment case.

Table 3. Doppler coefficients for UO_2 fuel and different Uranium enrichments. Uncertainties are expressed in pcm.

wt. (%)	CXS				DBRC			
	MCS		GREAPMC		MCS		GREAPMC	
	FTC	σ	FTC	σ	FTC	σ	FTC	σ
0.711	-4.640	0.015	-4.659	0.015	-5.128	0.014	-5.144	0.016
1.6	-3.026	0.010	-3.049	0.011	-3.401	0.010	-3.386	0.011
2.4	-2.560	0.009	-2.558	0.009	-2.845	0.009	-2.857	0.009
3.1	-2.339	0.008	-2.324	0.008	-2.609	0.008	-2.599	0.009
3.9	-2.152	0.007	-2.167	0.008	-2.410	0.008	-2.426	0.009
4.5	-2.064	0.007	-2.065	0.008	-2.309	0.007	-2.316	0.008
5.0	-2.017	0.007	-2.000	0.008	-2.261	0.008	-2.253	0.008

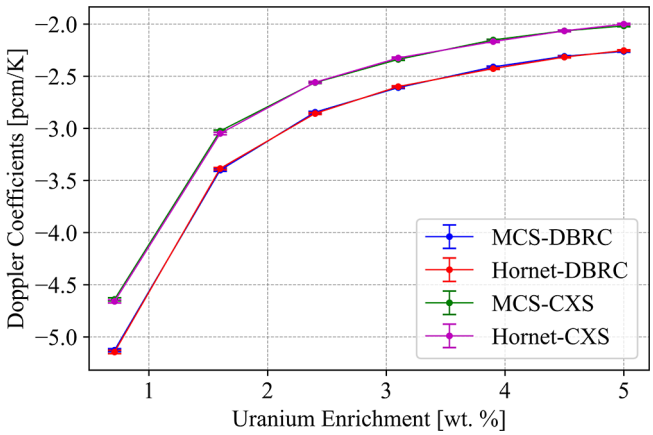

Figure 15. Doppler coefficients for UOX fuel with different Uranium enrichments.

Table 4. Doppler coefficients for reactor-recycle MOX fuel with different Plutonium enrichments. Uncertainties are expressed in pcm.

wt. (%)	CXS				DBRC			
	MCS		GREAPMC		MCS		GREAPMC	
	FTC	σ	FTC	σ	FTC	σ	FTC	σ
1	-3.610	0.011	-3.590	0.012	-3.979	0.012	-3.984	0.012
2	-3.462	0.012	-3.463	0.011	-3.812	0.012	-3.816	0.012
4	-3.294	0.010	-3.294	0.010	-3.611	0.011	-3.629	0.010
6	-3.160	0.009	-3.159	0.010	-3.464	0.010	-3.459	0.010
8	-3.041	0.009	-3.037	0.010	-3.318	0.009	-3.321	0.010

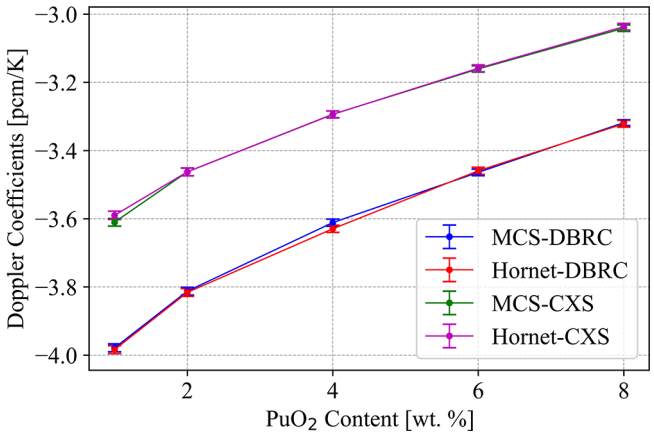

Figure 16. The Doppler coefficient for reactor-recycle MOX fuel steadily reduces in magnitude as Plutonium enrichment increases.

Table 5. Doppler coefficients for weapons-grade MOX fuel with different Plutonium enrichments. Uncertainties are expressed in pcm.

wt. (%)	CXS				DBRC			
	MCS		GREAPMC		MCS		GREAPMC	
	FTC	σ	FTC	σ	FTC	σ	FTC	σ
1	-2.553	0.009	-2.561	0.010	-2.883	0.009	-2.871	0.010
2	-2.633	0.008	-2.634	0.009	-2.906	0.008	-2.918	0.009
4	-2.611	0.008	-2.609	0.009	-2.883	0.008	-2.873	0.008
6	-2.519	0.008	-2.515	0.008	-2.744	0.007	-2.741	0.008

Figure 17 illustrates the Doppler coefficients for the weapons-grade MOX fuel at various plutonium enrichments. Both MCS and GREAPMC showed excellent agreement.

4.4 Pin-by-Pin power

OPR1000 is a commercial Korean reactor.[62] The peculiarity of the OPR1000 core is the guide tube and instrumentation tube. Although the fuel assemblies in most reactors have a uniform lattice, the OPR1000 fuel assemblies have a non-uniform lattice in the sense that four normal pin cells are combined to form one guide tube. Similarly, the instrumentation tube was larger than the normal pin cells.

The pin-by-pin power was quantitatively assessed by comparing the normalized pin powers in an OPR1000 fuel assembly (FA). Figure 18 shows the FA design loaded with 1.43% UO_2. The radial boundary condition is reflective, whereas the

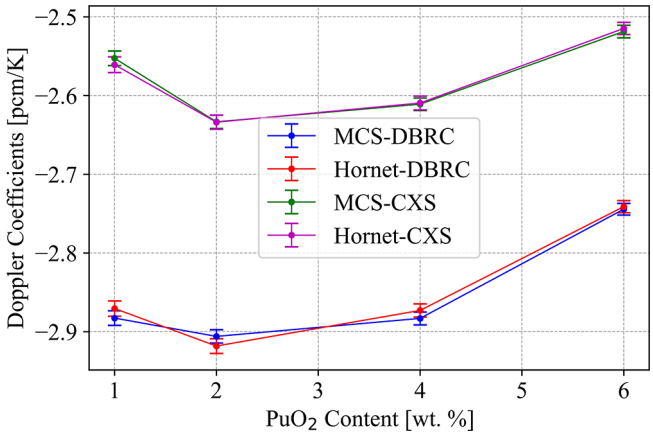

Figure 17. FTC for weapons-grade MOX fuel.

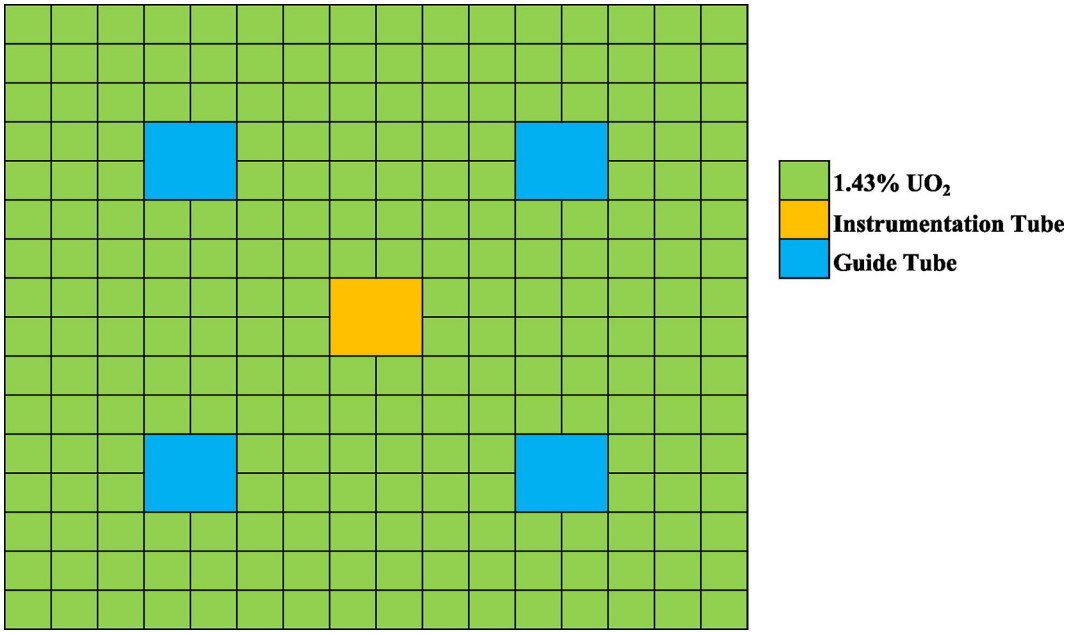

Figure 18. The 1.43% UO_2 FA design was used to assess the pin powers.

axial boundary condition is vacuum. Both the MCS and GREAPMC simulations were performed with 100 inactive cycles, 500 active cycles, and two million particles per cycle. The difference in eigenvalue computed using Equation (7) is 3.9±2.65 pcm. Figure 19 depicts the normalized radial pin power profile in quarter FA from both codes along with the relative difference, which is computed using Equation (9).

$$\text{Relative difference}\,[\%] = 1 - \frac{S}{R} \times 100 \tag{9}$$

The root mean square (RMS) difference in the axially averaged power profile was less than 0.1%, and the maximum relative difference was less than 0.2%. Overall, the power profile showed excellent agreement, as indicated by the RMS.

4.5 Performance

The performance of codebase was assessed for the OPR1000 FA case. Based on the codes' average time to execute one active cycle, GREAPMC's acceleration is compared with that of MCS. The number of particles per cycle varied from 1 million to 5 million. Constant cross-section approximations for elastic scattering and equi-lethargy-based energy lookup were employed for these simulations. Table 6 lists the hardware specifications. The MCS simulations utilized

	1.0773 1.0780 *-0.0615%*	1.0159 1.0151 *0.0829%*	0.9930 0.9932 *-0.0267%*	0.9807 0.9807 *-0.0013%*	0.9730 0.9729 *0.0040%*	0.9727 0.9725 *0.0142%*	0.9896 0.9896 *0.0026%*
1.0778 1.0778 *0.0057%*	1.0359 1.0358 *0.0145%*	1.0159 1.0152 *0.0785%*	1.0114 1.0114 *0.0043%*	1.0033 1.0030 *0.0338%*	0.9878 0.9877 *0.0135%*	0.9816 0.9813 *0.0245%*	0.9952 0.9946 *0.0617%*
1.0152 1.0152 *-0.0017%*	1.0155 1.0156 *-0.0185%*	1.0343 1.0335 *0.0762%*	1.0736 1.0723 *0.1197%*	1.0692 1.0686 *0.0516%*	1.0218 1.0216 *0.0157%*	0.9979 0.9973 *0.0544%*	1.0050 1.0051 *-0.0125%*
0.9935 0.9932 *0.0314%*	1.0111 1.0118 *-0.0633%*	1.0729 1.0726 *0.0360%*			1.0718 1.0715 *0.0271%*	1.0152 1.0148 *0.0415%*	1.0142 1.0131 *0.1017%*
0.9810 0.9808 *0.0179%*	1.0035 1.0033 *0.0183%*	1.0691 1.0686 *0.0485%*			1.0752 1.0747 *0.0462%*	1.0188 1.0186 *0.0219%*	1.0181 1.0176 *0.0505%*
0.9715 0.9727 *-0.1270%*	0.9874 0.9879 *-0.0523%*	1.0216 1.0217 *-0.0112%*	1.0717 1.0715 *0.0236%*	1.0750 1.0749 *0.0084%*	1.0326 1.0319 *0.0654%*	1.0103 1.0107 *-0.0349%*	1.0185 1.0187 *-0.0174%*
0.9723 0.9720 *0.0296%*	0.9813 0.9818 *-0.0501%*	0.9975 0.9973 *0.0164%*	1.0149 1.0149 *-0.0005%*	1.0185 1.0191 *-0.0527%*	1.0102 1.0108 *-0.0636%*	1.0077 1.0079 *-0.0213%*	1.0225 1.0230 *-0.0433%*
0.9890 0.9888 *0.0210%*	0.9945 0.9950 *-0.0549%*	1.0042 1.0045 *-0.0371%*	1.0139 1.0136 *0.0336%*	1.0180 1.0180 *0.0023%*	1.0186 1.0185 *0.0063%*	1.0228 1.0227 *0.0098%*	1.0418 1.0417 *0.0124%*

MCS
GREAPMC
Rel. Diff. (%)

Rel. Diff. (%)	
Min	-0.1270%
Max	0.1197%
RMS	0.0455%

Figure 19. Normalized pin powers and related statistics in quarter OPR1000 FA.

Table 6. Hardware specifications for the performance assessment.

GPU	Model	NVIDIA GeForce RTX 3090
	Global Memory (GB)	23.7
CPU	Model	Intel Xeon Gold 6242R
	Base Clock (MHz)	3100
	Memory (GB)	754
	Number of cores per node	40
	Number of nodes used for simulation	04

144 CPU cores, whereas the remaining 16 cores within the four nodes were reserved for development and administrative tasks.

The performance was quantified using the equivalent number of CPU cores that yielded the same performance as that of one GPU. The parameter is given by Equation (10)[63]:

$$\text{Equivalent Number of CPU Cores} = \frac{(\text{Number of CPU Cores}) \times (\text{CPU run time})}{\text{GPU run time}}. \quad (10)$$

In terms of tracking rate, the equivalent number of CPU cores is specified by Equation (11).

$$\text{Equivalent Number of CPU Cores} = \frac{(\text{Number of CPU Cores}) \times (\text{GPU tracking rate})}{\text{CPU tracking rate}}. \quad (11)$$

Table 7 presents the tracking rates for both codes. The tracking rate increases as the number of particles per cycle increases, with GREAPMC showing a more significant growth in the tracking rate than MCS. Consequently, the equivalent number of CPU cores increased accordingly.

Table 7. Performance of GREAPMC compared to MCS for the fuel assembly case.

Particle per cycle (Million)	Tracking rate (kn/s)		Equivalent number of CPU cores
	MCS	GREAPMC	
1	218.20	642.64	324
2	219.83	665.56	435
3	221.07	670.54	437
4	221.78	677.73	440
5	222.22	680.33	441

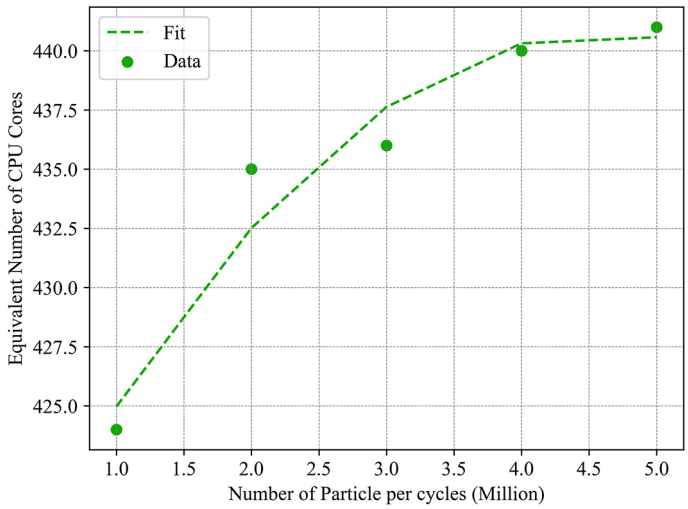

Figure 20. Equivalent number of CPU cores for the OPR1000 fuel assembly.

Figure 20 illustrates the equivalent number of CPU cores. As the number of particles per cycle increases, the equivalent number of CPU cores stabilizes at approximately 440, indicating that a single GPU provides a performance comparable to that of approximately 440 Intel Xeon Gold 6242R CPU cores.

5. Conclusions

The verification of Hornet against the well-established MCS code demonstrates its capability to deliver accurate and efficient solutions using GPU architecture. Hornet coupled with GREAPMC showed excellent agreement with MCS across a range of simulations. The implementation uses single-precision nuclear data, incorporates a variety of energy lookup methods, and adheres to an object-oriented programming paradigm, thereby ensuring code readability, maintainability, and adaptability. The results from benchmarks, such as the IAEA benchmark INDC (USA)-107 for a pin-cell model and the Mosteller benchmark for Doppler coefficient evaluations, confirm that Hornet achieves high precision across various nuclear conditions, including bound and free scattering scenarios. The pin-by-pin power verification in an OPR1000 fuel assembly further validated Hornet's performance with root-mean-square (RMS) power differences under 0.1%, highlighting its potential for accurate simulations. Combined, Hornet and GREAPMC significantly enhanced the tracking rate, surpassing that of the MCS. Overall, Hornet offers a robust solution for continuous-energy Monte Carlo neutron transport in GPUs, providing a valuable tool for nuclear analysis and reactor design. The future involves expanding Hornet's capabilities with photoatomic ACE file processing.

6. Declaration of generative AI in scientific writing

During the preparation of this work, the author(s) used ChatGPT-3 to improve readability, clarity, and conciseness of writing. After using this tool/service, the author(s) reviewed and edited the content as needed and took (s) full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this study.

Credit authorship contribution statement

Muhammad Rizwan Ali: Main developer, methodology, formal analysis, writing – original draft. Murat Serdar Aygul: Programming, result accumulation, review. Deokjung Lee: Supervision, fund acquisition, and review.

Software availability

The codes used in this study (Hornet and GREAPMC) are proprietary, in-house software developed by the CORE Lab at UNIST and are not publicly available due to intellectual property restrictions and university policy. Researchers interested in collaborating or seeking access to the software may contact the corresponding author.

Data availability

Some of the data supporting the findings of this study were generated using in-house codes developed at the CORE Lab, UNIST, and are subject to intellectual property restrictions. Therefore, the datasets cannot be shared publicly. No Institutional Review Board (IRB) approval was required for data sharing. Interested researchers may contact the corresponding author to request access to the input files associated with the non-proprietary models (like Mosteller Benchmark); requests will be considered on a case-by-case basis and are subject to CORE Lab and university approval.

Acknowledgment

This work was supported by an Innovative Small Modular Reactor Development Agency grant (RS-2023-00265742) funded by the Korean government.

References

1. Walsh JA, Romano PK, Forget B, *et al.*: **Optimizations of the Energy Grid Search Algorithm in Continuous-Energy Monte Carlo Particle Transport Codes.** *Comput. Phys. Commun.* Nov. 2015; **196**: 134–142.
Publisher Full Text
2. Ali MR, Aygul MS, Lee D: **Enhancing PWR Monte Carlo Simulations with GREAPMC: A GPU-Accelerated Approach.** *International Conference on Physics of Reactors (PHYSOR 2024).* San Francisco, CA, USA: pp. 2174–2183.
Publisher Full Text
3. Dzianisau S, Lee D: **GPU Acceleration of 3D MOC Solver in STREAM3D Using OpenACC.** *presented at the Transactions of the Korean Nuclear Society Spring Meeting, Jeju, Korea.* May 2024.
4. Carter Edwards H, Trott CR, Sunderland D: **Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns.** *J. Parallel Distrib. Comput.* Dec. 2014; **74**(12): 3202–3216.
Publisher Full Text
5. Medina DS, St.-Cyr A, Warburton T: **OCCA: A Unified Approach to Multi-Threading Languages.** *arXiv.* 2014; **abs/1403.0968**.
6. Beckingsale DA, *et al.*: **RAJA: Portable Performance for Large-Scale Scientific Applications.** *presented at the 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC).* Nov. 2019; pp. 71–81.
Publisher Full Text
7. OpenMP Architecture Review Board: **OpenMP Application Program Interface Version 5.0.** Accessed: Aug. 05, 2024.
Reference Source
8. OpenACC-standard. org: **The OpenACC Application Programming Interface Version 3.1.** Accessed: Aug. 05, 2024.
Reference Source
9. Reyes R, Brown G, Burns R, *et al.*: **SYCL 2020: More than meets the eye.** *Proceedings of the International Workshop on OpenCL. in IWOCL'20.* New York, NY, USA: Association for Computing Machinery; 2020.
Publisher Full Text
10. Stone JE, Gohara D, Shi G: **OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems.** *Comput. Sci. Eng.* Jun. 2010; **12**(3): 66–73.
PubMed Abstract | Publisher Full Text | Free Full Text
11. NVIDIA: **NVIDIA cuFFT.** Accessed: Aug. 05, 2024.
Reference Source
12. NVIDIA: **cuRAND - Random Number Generation on NVIDIA GPUs.** Accessed: Aug. 05, 2024.
Reference Source
13. NVIDIA: **cuSPARSE - GPU library APIs for sparse computation.** Accessed: Aug. 05, 2024.
Reference Source
14. NVIDIA: **cuBLAS - Basic Linear Algebra on NVIDIA GPUs.** Accessed: Aug. 05, 2024.
Reference Source
15. Conlin JL, Romano P: *A Compact ENDF (ACE) Format Specification.* Los Alamos, NM, United States, LA-UR-19-29016: Los Alamos National Laboratory (LANL); Sep. 2019.
Publisher Full Text
16. Brown DA: *ENDF-6 Formats Manual - Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI, ENDF/B-VII and ENDF/B-VIII.* Brookhaven National Laboratory, BNL-224854-2023-INRE; 2023.
17. Macfarlane R, Muir DW, Boicourt RM, *et al.*: *The NJOY Nuclear Data Processing System, Version 2016.* United States, LA-UR-17-20093: Jan. 2017.
Publisher Full Text
18. Tada K, Nagaya Y, Kunieda S, *et al.*: **Development and verification of a new nuclear data processing system FRENDY.** *J. Nucl. Sci. Technol.* Jul. 2017; **54**(7): 806–817.
Publisher Full Text
19. Los Alamos National Laboratory: **ACEtk.**
Reference Source
20. Belanger H: **Papillon Nuclear Data Library – A Free and Open-source C++/Python Library for Interacting with ACE Files for Continuous-Energy Neutron Data.** *EPJ Nucl. Sci Technol.* 2023; **9**.
Publisher Full Text
21. Romano PK, Horelik NE, Herman BR, *et al.*: **OpenMC: A State-of-the-art Monte Carlo Code for Research and Development.** *Ann. Nucl. Energy.* Aug. 2015; **82**: 90–97.
Publisher Full Text
22. Kowalski MA, Cosgrove P, Broman J, *et al.*: **SCONE: A Student-Oriented Modifiable Monte Carlo Particle Transport Framework.** *J. Nucl. Eng.* 2021; **2**(1): 57–64.
Publisher Full Text
23. Weiss MA: *Data Structures & Algorithm Analysis in C++.* Pearson Education; 4th ed. 2012.
24. Leppänen J: **Two Practical Methods for Unionized Energy Grid Construction in Continuous-Energy Monte Carlo Neutron Transport Calculation.** *Ann. Nucl. Energy.* Jul. 2009; **36**(7): 878–885.
Publisher Full Text
25. Leppänen J, Pusa M, Viitanen T, *et al.*: **The Serpent Monte Carlo code: Status, development and applications in 2013.** *Jt. Int. Conf. Supercomput. Nucl. Appl. Monte Carlo 2013 SNA MC 2013 Pluri- Trans-*

Discip. New Model. Numer. Simul. Paradig. Aug. 2015; vol. **82**: pp. 142–150.
Publisher Full Text

26. Martin WR: **Challenges and prospects for whole-core Monte Carlo analysis.** *Nucl. Eng. Technol.* Mar. 2012; **44**(2): 151–160.
Publisher Full Text

27. Lund A, Siegel AR, Forget B, *et al.*: **Using fractional cascading to accelerate cross section lookups in Monte Carlo particle transport calculations.** *presented at the Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method.* Nashville, Tennessee: Apr. 2015.

28. Cormen TH, Leiserson CE, Rivest RL, *et al.*: *Introduction to Algorithms, Third Edition.* The MIT Press; 3rd ed. 2009.

29. Liu Y, She D, Wang K, *et al.*: **Optimization treatment of point-wise nuclear data in Monte Carlo criticality and burnup calculations.** *Ann. Nucl. Energy.* Jul. 2011; **38**(7): 1489–1495.
Publisher Full Text

30. Brown FB: *New Hash-Based Energy Lookup Algorithm for Monte Carlo Codes.* Los Alamos National Laboratory, LA-UR-14-24530; 2014.

31. Wang Y, Brun E, Malvagi F, *et al.*: **Competing energy lookup algorithms in Monte Carlo neutron transport calculations and their optimization on CPU and Intel MIC architectures.** *J. Comput. Sci.* May 2017; **20**: 94–102.
Publisher Full Text

32. Raffuzzi V, Shwageraus E, Morgan L: **Accelerating Monte Carlo neutron transport by approximating thermal cross sections with functional forms.** *Ann. Nucl. Energy.* May 2022; **169**: 108819.
Publisher Full Text

33. Bergmann RM, Vujić JL: **Algorithmic Choices in Warp – A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs.** *Ann. Nucl. Energy.* Mar. 2015; **77**: 176–193.
Publisher Full Text

34. Scopatz A, Romano P, Wilson P, *et al.*: **PyNE: Python for nuclear engineering.** *presented at the 2012 ANS Annual Winter Meeting.* 2012; pp. 985–987.

35. Hamilton SP, Evans TM: **Continuous-Energy Monte Carlo Neutron Transport on GPUs in the Shift Code.** *Ann. Nucl. Energy.* Jun. 2019; **128**: 236–247.
Publisher Full Text

36. Choi N, Kim KM, Joo HG: **Optimization of Neutron Tracking Algorithms for GPU-Based Continuous Energy Monte Carlo Calculation.** *Ann. Nucl. Energy.* Nov. 2021; **162**: 108508.
Publisher Full Text

37. Choi N: *Development of GPU-Based Deterministic and Probabilistic Direct Whole-core Calculation Systems.* 서울대학교 대학원; 2021.
Reference Source

38. Tramm JR, *et al.*: **Toward Portable GPU Acceleration of the OpenMC Monte Carlo Particle Transport Code.** *presented at the International Conference on Physics of Reactors 2022 (PHYSOR 2022).* 2022.

39. Tramm J, *et al.*: **Performance Portable Monte Carlo Particle Transport on Intel, NVIDIA, and AMD GPUs.** *EPJ Web Conf.* 2024; **302**.
Publisher Full Text

40. Ridley G, Forget B: **Design and Optimization of GPU Capabilities In OpenMC.** *Presented at the 2021 ANS Winter Meeting.* 2021.

41. McKinley M, *et al.*: **Status of LLNL Monte Carlo transport codes on Sierra GPUs.** *Proceedings of the American Nuclear Society.* Portland, Oregon, USA: Aug. 2019.

42. Pozlup M, *et al.*: **Progress Porting LLNL Monte Carlo Transport Codes to Nvidia GPUs.** *Presented at the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering.* Niagara Falls, Ontario, Canada: Aug. 2023.

43. Morgan JP, Mote A, Pasmann SL, *et al.*: **The Monte Carlo Computational Summit – October 25 & 26, 2023 – Notre Dame, Indiana, USA.** *J. Comput. Theor. Transp.* Jul. 2024; **53**(5): 361–382.
Publisher Full Text

44. Descalle MA, Beck BR, Mattoon CM, *et al.*: **Implementation Of The GNDS Format For Evaluated And Processed Nuclear Data.** *Presented at the PHYSOR, Cancun, Mexico.* Apr. 2018.

45. NEA: *Specifications for the Generalised Nuclear Database Structure Version 2.0.* Paris: OECD Publishing; 2023.

46. Hawari AI: **On a measurement approach to support evaluation of thermal scattering law data.** *Ann. Nucl. Energy.* Jan. 2020; **135**: 106940.
Publisher Full Text

47. Tang Y, Zu T, Yi S, *et al.*: **Development and verification of thermal neutron scattering law data calculation module in nuclear data processing code NECP-Atlas.** *Ann. Nucl. Energy.* Apr. 2021; **153**: 108044.
Publisher Full Text

48. Fleming NC, Manring CA, Laramee B, *et al.*: **FLASSH 1.0: Thermal Scattering Law Evaluation and Cross Section Generation.** *EPJ Web Conf.* 2023; **284**.
Publisher Full Text

49. Squires GL: *Introduction to the Theory of Thermal Neutron Scattering.* Cambridge: Cambridge University Press; 3rd ed. 2012.
Publisher Full Text

50. Tran HN, *et al.*: **Comparison of the thermal neutron scattering treatment in MCNP6 and GEANT4 codes.** *Nucl. Instrum. Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip.* Jun. 2018; **893**: 84–94.
Publisher Full Text

51. Jiménez-Carrascosa A, Fridman E, Herranz N, *et al.*: **About the impact of the Unresolved Resonance Region in Monte Carlo simulations of Sodium Fast Reactors.** *Presented at the ICAPP 2019 - International Congress on Advances in Nuclear Power Plants.* Juan-les-pins, France: May 2019.
Publisher Full Text

52. Mosteller RD, Little RC: **Impact of MCNP Unresolved Resonance Probability-Table Treatment on Uranium and Plutonium Benchmarks.** *Presented at the ICNC'99: International Conference on Nuclear Criticality Safety.* Versailles, France: Sep. 1999.
Reference Source

53. Levitt LB: **The Probability Table Method for Treating Unresolved Neutron Resonances in Monte Carlo Calculations.** *Nucl. Sci. Eng.* Dec. 1972; **49**(4): 450–457.
Publisher Full Text

54. Sutton TM, Brown FB: **Implementation of the probability table method in a continuous-energy Monte Carlo code system.** *Presented at the International Conference on the Physics of Nuclear Science and Technology.* Long Island, NY (United States): Oct. 1998.
Reference Source

55. NVIDIA: **NVIDIA Compute Sanitizer.** Accessed: Aug. 05, 2024.
Reference Source

56. Menéndez ML, Pardo JA, Pardo L, *et al.*: **The Jensen-Shannon divergence.** *J. Frankl. Inst.* Mar. 1997; **334**(2): 307–318.
Publisher Full Text

57. Cullen DE, *et al.*: *How Accurately can we Calculate Thermal Systems?* United States, INDC (USA)-107: International Atomic Energy Agency; Apr. 2004.
Reference Source

58. Mosteller RD: *Computational Benchmarks for the Doppler Reactivity Defect.* Los Alamos, NM, LA-UR-06-2968: Los Alamos National Laboratory (LANL); Jun. 2006.

59. Mosteller RD: **ENDF/B-V, ENDF/B-VI, and ENDF/B-VII.0 Results for the Doppler-Defect Benchmark.** *Presented at the Joint International Topical Meeting on Mathematics and Computations and Supercomputing in Nuclear Applications.* California, USA: Apr. 2007.

60. Choi S, Lee H, Hong SG, *et al.*: **Resonance self-shielding methodology of new neutron transport code STREAM.** *J. Nucl. Sci. Technol.* Sep. 2015; **52**(9): 1133–1150.
Publisher Full Text

61. Sunny EE, Brown FB, Kiedrowski BC, *et al.*: **Temperature Effects of Resonance Scattering for Epithermal Neutrons in MCNP.** *Presented at the International Conference on Physics of Reactors (PHYSOR) - Advances in Reactor Physics.* Tennessee, USA: Apr. 2012.

62. Choi S, Lee D: **Preliminary Results for OPR1000/APR1400 Whole-core Analysis with Neutron Transport Code STREAM.** *Presented at the M&C.* Oregon, USA: Aug. 2019.

63. Hamilton SP, Slattery SR, Evans TM: **Multigroup Monte Carlo on GPUs: Comparison of History and Event-Based Algorithms.** *Ann. Nucl. Energy.* Mar. 2018; **113**: 506–518.
Publisher Full Text

Open Peer Review

Current Peer Review Status: ? ?

Version 1

Reviewer Report 16 September 2025

https://doi.org/10.21956/nuclscitechnolopenres.18966.r28232

?

Ilham Variansyah
Oregon State University, Corvallis, Oregon, USA

Overall Assessment
The manuscript presents the development and verification of Hornet, a GPU-optimized continuous-energy (CE) cross-section code, and demonstrates its integration within a GPU-based Monte Carlo code, GREAPMC. The work is timely, given the increasing interest in accelerating Monte Carlo (MC) reactor physics simulations through GPU architectures. The authors provide a solid overview of CE techniques in the context of GPUs, highlight their design choices (e.g., object-oriented paradigm, single-precision nuclear data, energy lookup strategies), and report extensive verification and performance results.

I find the paper suitable for indexing, and I recommend acceptance with some reservations. Most of my comments below are not critical revisions but are intended to help further strengthen the manuscript for clarity, completeness, and extended impact.

====

Context and comparisons to existing work
- The manuscript claims novelty in developing GPU-optimized CE cross-section processing that is Monte Carlo agnostic. However, this statement should be carefully nuanced in light of MCGIDI, which also functions as a standalone CE processing tool. A more direct comparison to MCGIDI (or MCGIDI-like approaches) would provide a stronger context for evaluating Hornet's unique contributions.
- Discussions on GPU-specific continuous-energy treatments in several Monte Carlo codes are given. It may also be worthwhile to discuss CE treatments in GUARDYAN for completeness.
- While running a full MC transport simulation using the GREAPMC-Hornet code system provides a useful performance comparison, tools/benchmarks such as XSBench might serve as a more direct baseline for assessing computational performance of CE physics alone.
- Other than enabling multigroup GPU MC code (like GREAPMC) to run continuous-energy physics and serving as an academic tool, authors may also want to discuss potential uses of Hornet-like codes in a CPU MC code, such as an asynchronous cross-section lookup that has been explored in

the MC code MC21.

Hornet VS GREAPMC-Hornet evaluations
- Authors should emphasize that most of the presented verification exercises and performance assessments are those of the GREAPMC-Hornet system, not the Hornet code itself. The emphasis is important because verification exercises for the GREAPMC-Hornet system do not necessarily verify Hornet itself (integration VS unit verification). The same goes for the performance assessment. Nevertheless, evaluation of the GREAPMC-Hornet system is still relevant and important to the paper, as it is one of the main purposes of the Hornet development and a typical use of such a modular code.
- Consider tools/benchmarks like XSBench for assessing the computational performance of CE physics alone.
- The authors should briefly discuss any changes in the Hornet-enabled GREAPMC from the cited, original multigroup GREAPMC. If a separate paper will be published to provide an in-depth discussion of this, please state so.

Outgoing energy distribution sampling comparison
- The use of Jensen–Shannon divergence (JSD) to quantify agreement between MCS and GREAPMC-Hornet sampled distributions is innovative. However, most readers probably will not be familiar with the interpretation of JSD values. Please discuss the typical range or min/max of JSD values and what constitutes “small” and whether the value of 0.1 can be considered sufficiently low in this context.
- Certain incident energies and temperatures are used for generating the results. Please state whether similar agreements are observed in other energies and temperatures.

Verification through code-to-code comparison of MC results
The presented verification effort compares particular instances of MC simulations (at a given number of samples N). To better verify through code-to-code comparison of MC results, one can demonstrate whether the two codes' results would reasonably converge to the same answer at the expected rate of 1/sqrt(N). Consider running with (logarithmically) increasing numbers of samples N and observe the convergence of the relative difference (average of the two can be used as the reference), and take the norm/RMS of the differences to get a single measure for each N. Note that this would also shed light on the significance of the single vs double-precision arithmetic, which would be valuable. Similar error convergence analysis can be used in the Thermal Systems, Mosteller Benchmark, and Pin-by-Pin Power verifications as well.

Content Accuracy and Verification
- "Most of the runtime in the Monte Carlo (MC) code is spent on continuous-energy cross-sectional lookups." How universal the sentence is is debatable. Consider "In many classes of MC transport simulations, particularly neutron fission reactor systems,". Note that systems with highly complex geometry, such as fusion neutronics and medical imaging, can be counterexamples to this.
- For transparency, it would be helpful to state explicitly whether all numbers of inactive cycles used are sufficient to ensure fission source convergence.
- Please state whether the uncertainties in the keff difference in the Thermal Systems are propagated standard deviations.
- Please briefly discuss how the standard deviations in the Doppler coefficients results (tables and figures) are calculated/propagated, particularly on how the possible correlation is treated or avoided.

Performance Assessment
- The GPU-vs-CPU performance comparison properly involves increasing workload to demonstrate saturation of parallel computing efficiency on both GPU and CPUs.
- Performance metrics reported are based on active cycle runtime. Please state whether the runtime assessment excludes cycle closeout operations, such as tally score accumulations and fission bank management for reproducibility.
- Please discuss what tallies are included in the performance calculations. Would the size of the tallies affect the performance comparison results?
- Please explain why certain physics treatments (e.g., constant cross-section elastic scattering, equi-lethargy energy lookup) were used in the performance tests. How sensitive are the results to these treatment choices, and are they particularly favorable to GPU execution?
- Please clarify whether the development/administrative tasks running on 10% of cores (16 total) had any measurable impact on the MCS performance assessment. If negligible, this should be mentioned along with steps/treatments taken to isolate the MC computations.
- A note on whether MCS also uses shared-memory parallelism, and (if it does) whether a single-node run would improve MCS parallel efficiency would be useful.

====

Minor Comments

Wording
- “the continuous treatment … reduces the errors introduced by discretization” --> “the continuous treatment … avoids the errors introduced by discretization”.
- “Microscopic cross sections are the interaction probability” --> “Microscopic cross sections measure/quantify the interaction probability”.

Equations
- Equation (2): Consider denoting macro cross sections with Σ symbol to distinguish them from summation signs.
- Equation (9): Appears to be missing parentheses.

Formatting
- Section 2.1.5 (Hash-based energy lookup) heading should be bolded for consistency.
- Correct the fragment “However, the evaluated nuclear data Files) data”.

References
- Please provide a proper bibliographic citation for Chen et al. (2018) instead of only a URL.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?

Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Radiation transport, Monte Carlo method, high-performance computing, reactor physics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Reviewer Report 05 September 2025

https://doi.org/10.21956/nuclscitechnolopenres.18966.r28231

?

Ho Jin Park
Kyunghee University College of Engineering, Yongin-si, Gyeonggi-do, South Korea

In this paper, the authors introduced new continuous-energy cross section processing code system, Hornet for GPU-based Monte Carlo simulations. This Hornet code can generate the ACE format cross section and efficiently handle the cross section data for GREAPMC code. The tables and figures are well-organized to effectively support the results. I have some questions and comments before publication.

1. In many GPU/CPU-based Monte Carlo analysis codes, routines for reading or processing nuclear data are naturally included. However, the CUDA-based Monte Carlo code developed by the authors separates this function and is divided into two codes, Hornet and GREAPMC. In this manuscript, sufficient justification and rationale for this approach are not provided.
2. If you wish to publish on this system, I recommend doing so with a manuscript that covers the integrated system combining GREAPMC and Hornet. Based on the content of the current paper, Hornet does not appear to have any distinctive features compared to conventional Monte Carlo cross-section processing. To adequately address the originality of this research, you should either emphasize such distinctive features or submit the work together with the development details of GREAPMC.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Reactor Phyics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.