EPJ Nuclear Sci. Technol. **11**, 62 (2025)

https://doi.org/10.1051/epjn/2025053

EPJ Nuclear Sciences & Technologies

Special Issue on 'Overview of recent advances in HPC simulation methods for nuclear applications', edited by Andrea Zoia, Elie Saikali, Cheikh Diop and Cyrille de Saint Jean

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

REGULAR ARTICLE

OPEN ACCESS

Accelerating split-exponential track length estimator on GPU for Monte Carlo simulations

Henri Hutinet[1,*], Pierre-Louis Antonsanti[1], Cindy Le Loirec[1], and Davide Mancusi[2]

[1] CEA, DES, IRESNE, DER, Service de Physique des Reacteurs et du Cycle, Cadarache, F-13108 Saint-Paul-lez-Durance, France

[2] CEA, Université Paris-Saclay, Service d'Etudes de Réacteurs et de Mathématiques Appliquées, 91191 Gif-sur-Yvette, France

Received: 31 May 2025 / Received in final form: 5 August 2025 / Accepted: 7 August 2025

Abstract. In the context of computing 3D volumetric tallies for nuclear applications, the combination of Monte Carlo methods and high-performance computing is essential to achieve accurate yet computationally feasible simulations that meet industrial time constraints. The next-event *Split Exponential Track-Length Estimator* (*se*TLE) is particularly well suited for estimating tallies on meshes. To alleviate the computational burden associated with *se*TLE, such as sampling numerous outgoing pseudoparticles at each collision, estimating cross sections, performing ray tracing through complex geometries, and accumulating scores across the geometry, we leverage the parallel computing capabilities of Graphics Processing Units (GPUs). We assess the performance of our implementation using two shielding configurations and one criticality benchmark. Both photon and neutron transport simulations are considered. Scores are evaluated over Cartesian meshes, material volumes, and energy group structures. In all cases, acceleration factors greater than unity are observed in the detectors, reaching several hundred in selected regions of the phase space. In a final experiment, we demonstrate that our GPU-based implementation achieves a net energy gain (in Watt) even when compared to a conventional CPU-based TLE, despite the additional computational cost of GPU use.

1 Introduction

The behavior of neutral particles, such as neutrons and photons in matter, is typically described using the Boltzmann equation. Monte Carlo simulations solve this equation without introducing any approximations to the method, making it the gold standard in nuclear engineering [1], especially for mapping observables of interest (tallies) in 3 dimensions. Applications include the monitoring of reactor cores such as in-core fluence for local power densities and reaction rates or the preparation of decontamination and dismantling (D&D) operations (estimation of neutron flux for activation of reactor structures, or equivalent dose rates for radiation protection). Such 3D calculations are often performed with volumetric estimators: the collision estimator [2] for the analog random walk counting the number of interactions in each volume of interest, or the Track Length Estimator (TLE) [3], recording the intersection between volumes and particles' trajectories during analog random walks. They are both unbiased but exhibit different convergence rates depending on the application scenarios [1]. The former is not well adapted to volumes with optically thin material. The latter, namely the TLE, typically yields lower variance than the collision estimator, especially when the traversed medium is optically thin. It is the default estimator in MCNP (tally F4) [4]. However it slowly converges everywhere in 3D maps. This behavior is explained by the central limit theorem, which states that the convergence speed of estimators – random variables – scales with the square root of the sample size. This challenge becomes even more pronounced in radiation protection calculations, where substantial attenuation – often spanning several orders of magnitude – occurs between the source and the detection region. Over the years, several methods have been developed to address these challenges. A wide range of algorithmic strategies, commonly referred to as variance reduction techniques (VRTs) [5], have been developed, with most of them utilizing importance sampling [6]. Among these, the most widely adopted and standard method in most Monte Carlo codes are based on the concept of statistical weights, using population control methods like splitting and Russian roulette for implicit capture. For radiation shielding calculations, more advanced techniques are employed that leverage importance maps. The two predominant methods in this domain are the Exponential Transform

* e-mail: henri.hutinet@cea.fr

[7] and the Weight Windows method [8]. In criticality simulations, the delta-tracking method has been extensively refined [9]. This method bypasses particle tracking within the geometry by employing a rejection algorithm based on the computation of a major macroscopic cross-section.

Using an appropriate estimator can also accelerate the convergence of tallies. A more advanced estimator, known as the exponential track-length estimator (eTLE), can be derived from the TLE formulation. First introduced by Williamson in 1987 [10], the key distinction between these two estimators lies in replacing a stochastic decision – namely, the free flight of a particle – with its expected outcome, averaged over the probability distribution of particle flight lengths. Specifically, eTLE operates by deterministically transporting "pseudo" or "virtual" particles, stochastically created at each emission and collision of a "real" particle, all the way to the boundary of the geometry. Consequently, this method engages the two most computationally intensive tasks in Monte Carlo methods: cross-section interpolation and geometric computations. In general, for the same number of simulated particles, eTLE achieves lower variance than TLE, albeit at the cost of increased computational time [1,11]. Another well-known expectation technique is the forced-flight DXTRAN method from MCNP [12], which follows a similar principle. Likewise, the next-event point estimator applies this approach to calculate tallies at specific points in phase space [13].

Hardware-based approaches can also be used to improve simulation efficiency. They often focus on massively parallelizing simulations, making them highly suitable for Monte Carlo transport. Over the past decade, advances in scientific computing on GPUs have also been applied to the Monte Carlo method, yielding promising results [14]. The graphical processing power of personal computers has significantly increased, as the GPU share in the world's leading supercomputers. Notable examples include the ORNL OLCF-5 exascale supercomputer, Frontier [15], where each node consists of one CPU and four GPUs. In France, the Tera1000-1 supercomputer features 400 GPUs dedicated to visual rendering and high-performance scientific computing [16]. Given this shift, the Monte Carlo method for particle transport must evolve to align with the new GPU parallelism paradigm. A conference report [14] on Monte Carlo radiation transport outlines the state of the art for major American codes adapted for GPU computing. Two main trends have emerged. In the first approach, the GPU handles most tasks related to particle transport. Most Monte Carlo codes running on physical CPU cores employ a history-based tracking method, where the random walk of particles is simulated sequentially. A straightforward implementation of this approach on a GPU would involve assigning multiple particles to a kernel, with each thread simulating a particle's trajectory. However, the numerous conditional branches inherent in such simulations cause thread divergence, exacerbating the tail-effect. To address this issue, the community has revisited the event-based tracking algorithm proposed by Brown and Martin in the 1980s [17], which is better suited for GPU parallelization. This technique is implemented in codes like WARP [18], GUARDYAN [19], Shift [20], PRAGMA [21] and Patmos mock-up at CEA [22], each with its own specific features. The second approach employs a hybrid CPU/GPU transport algorithm, where the particle random walk is managed on the CPU using a history-based method, while certain calculations are executed asynchronously on the GPUs. This strategy minimizes the impact on the often well-established Monte Carlo codes and addresses the limitations of fully porting these codes to GPUs, such as constrained memory capacity. For instance, [23] propose performing cross-section calculations (energy bin lookup and interpolation) on a GPU.

Many of these methods can be combined to achieve faster convergence that meets the user's requirements. In this context, we propose a method that integrates the *exponential Track-Length Estimator* (eTLE) and partial GPU implementation. Recently, Guadagni implemented eTLE in TRIPOLI-4® [24,25] for photon transport in its simplest form, combining it with a forced-flight method [26], inspired by Cramer [5]. This study demonstrated the efficiency of the estimator when paired with the forced-flight method. Subsequently, Hutinet adapted these techniques for neutron transport in TRIPOLI-4® [27], showcasing the effectiveness of eTLE with Forced Detection in shielding problems. This approach was proven superior to the traditional TLE, both with and without VRT based on importance maps [28]. The eTLE has further evolved to handle multiple directions at each source or collision sampling by emitting several pseudo-particles instead of just one. This variant, known as the *Split Exponential Track-Length Estimator* (seTLE) [26,29,30], introduces additional computational complexity. The transportation of these pseudo-particles through boundary geometries imposes significant performance constraints on expectation estimators. Consequently, for the same number of simulated particles, the reduction in variance compared to TLE is counterbalanced by increased simulation times. To overcome this limitation, we propose harnessing the power of graphics processors to perform ray tracing of pseudo-particles asynchronously. Since straight-line pseudo-particle transport through geometry is perfectly suited to parallelization, it can be efficiently executed on GPUs. Several authors have already explored porting estimators to GPUs to take advantage of this [31,32].

In this paper, we propose to port the seTLE recently implemented in TRIPOLI-4® [26,28] to GPU. We adopt the point of view of [32] and use a hybrid CPU/GPU architecture by offloading ray tracing to GPU. The work presented here aims to build upon the main principles of the method and adapt them to various needs. First, the implementation is designed to use exact geometries rather than voxelized ones, in particular allowing for multiple materials per cell. The method is subsequently extended for application to both criticality and fixed-source problems involving neutrons or photons, and can provide fluence, H*10 dose equivalent rate and main reaction rates. An algorithm for sampling the particle scattering parameters – namely, energy and direction – is also developed on the GPU to enable full implementation of the method on the graphics card. The algorithm is subjected to a

parametric study and compared to the traditional sampling approach executed on CPU cores. We also propose performing score aggregation on the GPU at each batch end to further offload the CPU physical cores. This approach is particularly well suited for a large number of scores, as all GPU's threads perform the same operation. Finally, the paper presents an additional sensitivity analysis on the refinement of tallies within phase space (spatial and energy domains) to better characterize the estimator's behavior. To ensure a fair comparison, an energy-based evaluation is conducted between the CPU track-based and GPU seTLE-based methods.

The paper is constructed as follows: in Section 2, we discuss the method, and detail the different steps involved in the GPU implementation of *se*TLE: sampling, ray tracing, and accumulation of the scores across the batches with respect to the cells of the mesh. In Section 3, we quantify the gain of each step, the influence of the different parameters of the method on the simulation performances, which we illustrate with three examples. We conclude and discuss future development in the last section.

2 Method

The underlying idea of this work is to port the maximum number of algorithmic components associated with the *se*TLE calculation to the GPU. In this way, the CPU is left only to perform the calculations associated with the random walk of the simulated particles.

2.1 Workflow overview

We describe in Figure 1 the workflow of the adaptation of *se*TLE on GPU. As mentioned in the last section, the *se*TLE is based on straight-line pseudo-particle transport in the geometry. These pseudo-particles are generated at each collision of a particle following the random walk. The random walk is performed on CPU as usual, and the left-hand side (blue) of the diagram represents this random walk. Let $P = (r, \Omega_{\text{in}}, E_{\text{in}})$ be a pseudo-particle sampled (one sampling corresponds to the *e*TLE). If the ray (r, Ω_{in}) intersects a homogenous detector d, of total macroscopic cross-section $\Sigma_T^d(E_{\text{in}})$ at energy E_{in}, an unbiased estimator of the fluence given a pseudo-particle is:

$$\theta_{e\text{TLE}}^{(\phi,d)} = \exp\left(-\int_0^{\|\mathbf{r_0^d}-\mathbf{r}\|} \Sigma_t\left(\mathbf{r}+s\mathbf{\Omega}_{\text{in}}, E_{\text{in}}\right)\,\mathrm{d}s\right) \times \left(1-\exp\left(-\Sigma_t^d(E_{\text{in}})l\right)\right)\frac{w}{\Sigma_t^d(E_{\text{in}})} \tag{1}$$

with $\Sigma_t^d(r, E_{\text{in}})$ being the total cross-section at energy E_{in} and position r, r_0^d the first intersection between the volume detector frontier and the pseudo-particle ray and l the length of the chord traversed by the pseudo-particle in the detector. Equation (1) can be interpreted as follow: the first factor is the probability that the pseudo-particle interacts before the frontier of the volume detector; the second factor is the probability of interaction inside the detector and the third term is the usual collision estimator inside the detector. The product of the second and third terms correspond to the expected value of the collision estimator over all chord lengths in the detector. The main interest of this estimator is to contribute in detectors that are not necessarily crossed by real particles during the random walk. In the *e*TLE, the outgoing pseudo-particle results from the sampling of the random walk. The *se*TLE is the generalization of the *e*TLE to NB_{split} pseudo-particles instead of one, sampled independently with a corrected weight. The estimator thus becomes:

$$\theta_{se\text{TLE}}^{(\phi,d)} = \exp\left(-\int_0^{\|\mathbf{r_0^d}-\mathbf{r}\|} \Sigma_t\left(\mathbf{r}+s\mathbf{\Omega}_{\text{in}}, E_{\text{in}}\right)\,\mathrm{d}s\right) \times \left(1-\exp\left(-\Sigma_t^d(E_{\text{in}})l\right)\right)\frac{w}{NB_{\text{split}}.\Sigma_t^d(E_{\text{in}})} \tag{2}$$

with NB_{split} being called the splitting multiplicity. The sampling of the NB_{split} pseudo-particles at each collision is the central box of the Figure 1, and we adapted it to both CPU and GPU. This splitting routine is not applied to source sampling and particles resulting from boundary conditions for which a simple *e*TLE (Eq. (1)) is applied. Once the bank is filled, we cast the rays on GPU. A single thread on the device treats a single ray in a massively parallel way, similar to image rendering in computer graphics. The GPU-based *se*TLE has been written using CUDA C/C++ [33]. The code is also adapted to CPU parallelization by duplicating the problem: each physical CPU core involved performs a random walk, fills a buffer and sends it to the device; hence, the GPU memory footprint is proportional to the number of physical cores. The advantage is that CPU cores are not slowed down by filling the bank faster than the GPU can process it, but there is a risk of saturating the GPU memory and calculation capacity. In the following, we will detail each of the GPU steps for the computation of the *se*TLE. We may refer to the GPU as the device and to a physical CPU core as the host. Then, after introducing the configurations used in this work in Section 3.1, we provide a profiling analysis in Section 3.2 to motivate the choices of porting the described steps involved in the *se*TLE to GPU.

2.2 Outgoing particle parameters sampling

By default, implicit capture is active in TRIPOLI-4®. Three reactions are explicitly simulated for photons: the pair creation, Rayleigh scattering and Compton scattering (photoelectric emission results in implicit capture). For the first one, the photon is re-emitted isotropically with a doubled statistical weight. The second one is coherent; the photon's energy remains unchanged. The distribution of the cosine of the polar angle follows the Thomson differential cross-section weighted by the atomic form factor $F(q, Z)$ that depends on the momentum transfer q and the atomic number Z of the nucleus target. The form factors are tabulated. For the third reaction, the cosine of the scattering polar angle and the outgoing photon energy E_{out} can be sampled from the Klein-Nishina differential cross-section according to Brusa's fast algorithm

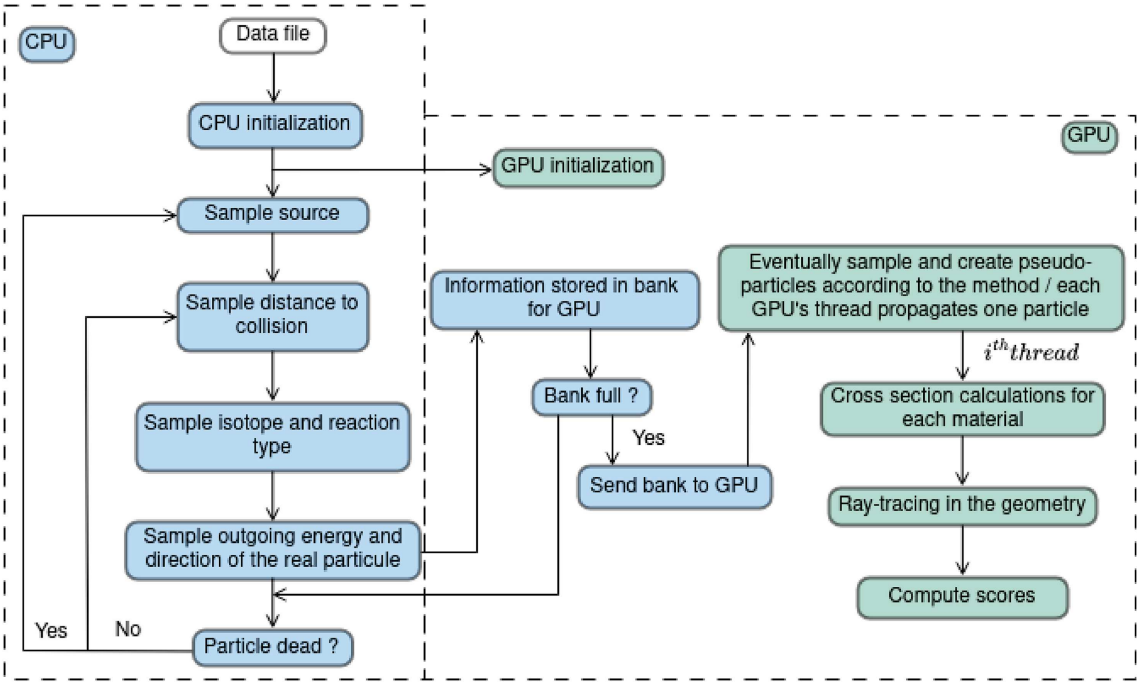

Fig. 1. GPU *se*TLE workflow.

[34]. For neutrons, TRIPOLI-4® transforms the probability density function of the cosine of the polar angle into equiprobable cosine bins to expedite the sampling for elastic scattering and discrete levels of inelastic scattering [24]. For the other reactions and according to the target, the energy and angular distributions of the emitted neutron can be independent or not. When input data are provided as tabulated quantities, different interpolation laws can be used in different ranges. At low energy, thermal neutron scattering law data are also available. The Sample Velocity Target (SVT) and Doppler Broadening Rejection Correction (DBRC) algorithms are employed to account for Doppler broadening of elastic scattering kernels in the epithermal region [35]. The entire information needed for the implementation of all the samplings on GPU is provided in the international evaluation files and transferred to the GPU. In this work, the algorithms used for sampling the return parameters are identical whether processed by a GPU thread or a physical CPU core.

2.2.1 (Ω, E) sampling

Sampling the outgoing particles parameters $(\Omega_{\text{out}}, E_{\text{out}})$ can be done either on the CPU or on the GPU. Using GPU sampling allows limiting the CPU's computations during the analog random walk simulation. All the other operations necessary to the *se*TLE are performed on GPU. The impact of the use of the *se*TLE on the simulation time is thus minimized. In return, the GPU computational capacity may saturate, and more information needs to be transferred to the GPU: supplementary nuclear data information at initialization and collision type of the real particle for each ray of the bank. The library used for random number generation is cuRAND. At initialization, each thread generates its sequence of random numbers on the device with the same seed but a different sequence number. The generator used (XORWOW) with a period of 2^{192} has good statistical properties [33].

On the contrary, CPU sampling reduces the amount of communications with the GPU and reduces the computational burden of the device. For instance, when using parallel CPU random walks communicating with a single GPU, CPU sampling may allow the use of more physical CPU cores. However, in particular, when sampling many pseudoparticles at a time, it comes at the cost of greatly increasing the simulation time. The choice of sampling the outgoing particles parameters depends on the hardware on which the user wants to place the computational cost. It also depends on the hardware itself, on the configuration of the simulation – for instance, on the number of collisions per particle history –, so the choice of using the sampling on GPU is left to the user.

2.2.2 Cross-sections calculation

Once the outgoing particles parameters have been calculated and before performing ray tracing in the geometry, it is necessary to calculate the total macroscopic cross-section of all materials according to the outgoing ray's energy, as required in Equation (1). Isotope microscopic cross-section lookup at a given energy is achieved with a binary search and interpolation. Then, the total macroscopic cross-sections of materials are computed. All materials cross-sections for each ray's energy are stored in a static vector on the device accessible during the ray tracing. The vector size is the bank size multiplied by the number of materials (NB_{mat}). For example, the element

$i^{\text{th}}.NB_{\text{mat}} + j^{\text{th}}$ gives the j^{th} material total macroscopic cross-section associated to the i^{th} ray's energy.

2.2.3 Ray tracing and mesh scoring

Once the bank is filled with outgoing pseudo-particles, the rays are cast in straight lines through the geometry, and they score in Cartesian grids or in volumes. We represent geometry with a limited number of primitives: spheres, boxes, and cylinders. For each ray cast, all the primitives are intersected on the GPU, and the ray history is computed. Following this history, the score derived from Equation (2) is then retrieved. The mesh for scoring is a Cartesian grid that can be non-uniform and irregular. The score in its cells are computed after the history of the rays. This implementation allows for exact ray/geometry intersection and the detectors (mesh grid cells) do not need to be filled with only one material. During the run of a batch of particles, the score in each cell is updated online on GPU using a shared access to the device memory through atomic operations. The update of the means and standard errors is done using Welford's online algorithm [36]. It is applied at the end of a batch, averages and standard errors are computed in parallel for each cell by GPU threads. This enables parallel scoring that is key when scoring in millions of detectors.

3 Results

The theoretical proof that the *se*TLE estimator is unbiased is given in [1]. In the different simulations, the GPU version of the *se*TLE has been compared, cell-wise, to the CPU version of the TLE. The comparison of the estimators is based on the distribution of the cell-wise Δ/σ difference:

$$\frac{\Delta}{\sigma} = \frac{\bar{x}_i - \bar{x}_j}{\sqrt{\sigma_i^2 + \sigma_j^2}} \tag{3}$$

with $\bar{x}_i, \sigma_i$ the mean value and its standard error estimated with a method i, and $\bar{x}_j, \sigma_j$ the mean value and its standard error estimated with a method j. In all the simulation, this score has a mean value over the mesh cells close to zero (0.01 in general) and the proportion of the cells follows 68%/95%/99% at 1/2/3 σ, respectively. This follows a normal distribution very closely; the *se*TLE GPU results are thus coherent with the reference implementation of the TLE. The unbiasness of the *se*TLE is not discussed in this paper. More details about this validation can be found here [28]. In the following, we will focus on the performances of the GPU implementations evaluated using a RTX A4500.

3.1 Monte Carlo configurations

Three configurations are used to assess the performance of the *se*TLE on GPU. The first configuration, illustrated in Figure 2, top-left, is a typical radiation protection situation in D&D: a concrete bunker with an internal wall, partially traversed by a cylindrical duct. A 1.33 MeV monokinetic, isotropic, point source of photons is used. A uniform Cartesian grid of size $100 \times 100 \times 100$ is used to compute the H*(10) map. In this simulation, 10^5 batches are performed with 2×10^4 particles per batch. In this configuration, the challenge lies in scoring behind the wall in the room. Two types of evaluation are therefore possible: either a global evaluation of all the cells in the grid, or an evaluation restricted to the cells in the room behind the wall.

The second configuration is a PWR 15×15-fuel assembly, illustrated in Figure 2, top-right. It consists of 209 UOX rods and 16 burnable poison tubes, for a total of 771 volumes. The assembly is 21.525 cm^3, and the Cartesian mesh is composed of $215 \times 215 \times 50$ cells. In this criticality simulation, 10^4 batches of 2×10^3 particles are performed, with 50 inactive cycles. We set reflection conditions at the boundaries of the geometry. An initiating source is placed at the center of the assembly. It is pointwise, isotropic and emits neutrons within the Watt spectrum.

The third configuration reproduces a classical benchmark problem widely employed in radiation shielding studies: the bent dogleg duct geometry, illustrated in Figure 2. This setup, adapted from [37], consists of three regions of interest. The first is an air-filled room (dimensions: 150 cm × 300 cm × 200 cm) in which the radiation source is located. This room faces a concrete-shielded bent duct system, which constitutes the second region. The bent duct has a square cross-section of 30 cm × 30 cm and traverses the shielding wall, emulating an auxiliary penetration through a reactor biological shield. The duct comprises three straight segments: the first and third are parallel, while the second is orthogonal to both. The lengths of the segments are 115 cm, 90 cm, and 65 cm, respectively. The third region, also filled with air and measuring 170 cm × 300 cm × 200 cm, is located downstream of the shielding structure. It represents a potentially occupied zone and is protected by the concrete barrier. The entire configuration is enclosed by 50 cm thick concrete walls. In this study, only volumetric tallies are used; no cartesian mesh tallies are employed. Eleven spherical detectors with a radius of 1 cm are positioned within the geometry: six along the duct and five in the third region, both behind and along the shielded wall. Two distinct simulations are performed. The first involves an isotropic photon source with two discrete emission energies corresponding to ^{60}Co decay (1.17 MeV and 1.33 MeV). The second considers an isotropic neutron source with monoenergetic emission at 14 MeV, characteristic of the fusion neutron emission spectrum. For each case, particle fluxes are recorded independently using both TLE and *se*TLE methods at all detectors.

3.2 seTLE computing time analysis on CPU

To motivate the use of the GPU for the computation of the necessary steps of the *se*TLE, we illustrate in Table 1 its computing time when implemented on CPU. To that end, we use the bunker configuration introduced in Figure 2, with 10^4 particles per batch. On average, the number of collisions per particle lifetime is 11, hence an average of $NB_{\text{split}} \times 11 \times 10^4$ pseudo-particles processed per batch. We

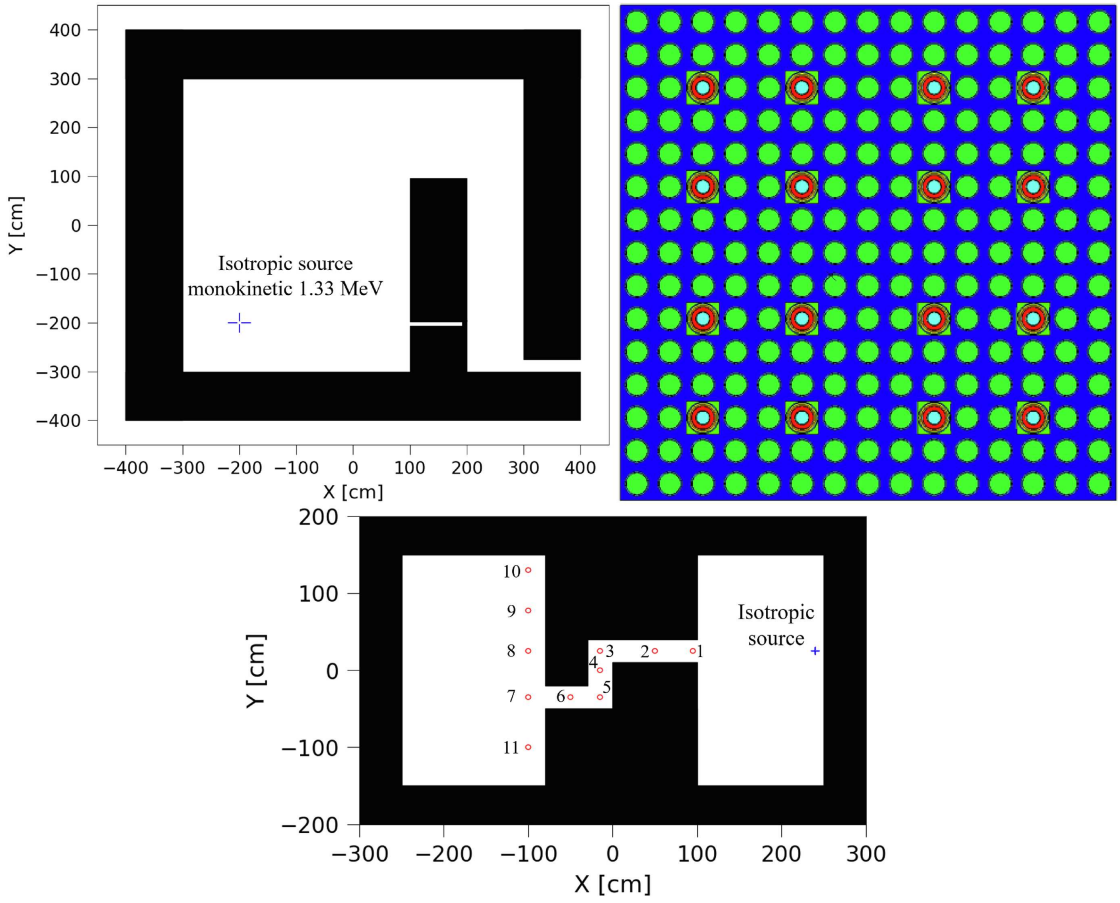

Fig. 2. (Top left) A bunker configuration in D&D. The performances of the *se*TLE GPU in terms of Figure of Merit ratio are assessed behind the concrete shield. (Top right) 15 × 15 UOX fuel assembly (21.525 cm^3) with 16 burnable poison tubes. (Bottom) Dogleg configuration with eleven detectors.

are interested in the average time for sampling (Ω, E), calculating the cross-sections and transporting the pseudo-particles across the geometry, relative to the average batch time using the TLE simulation on CPU.

One can notice that the computing time of each of the evaluated steps linearly increases with the splitting multiplicity. This is explained by the sequential processing of the particles and pseudo-particles on CPU. In addition, with this simple geometry of Figure 2 and considering $NB_{\text{split}} = 1$, the transport of the pseudo-particles takes the same amount of time as the complete batch simulation of the TLE. In other words, the use of *e*TLE on CPU at least double the simulation time for this configuration. Another observation that can be made is the high influence of the ray tracing across the geometry on the total computing time. For a splitting multiplicity of 32, it takes 100 times the total time of the batch simulation using TLE. The benefits of porting these calculations on GPU are thus twofold: first, it allows us to perform them asynchronously with the natural random walk performed on CPU (see Fig. 1). Second, each pseudo-particle can be treated independently in a massively parallel way and enabling splitting multiplicity higher than 1 by leveraging the GPU compute capabilities.

3.3 Performance of each step

The influence of porting the different steps on GPU is assessed using the bunker configuration. One can either evaluate the computation time to focus on the computing performance, or evaluate the simulation performance using the Figure Of Merit (FOM): FOM $= \frac{1}{t.\sigma^2}$, with t the physical CPU cores simulation time and σ the Monte Carlo simulation standard deviation. From the FOM, one can derive the speed-up factor of a given method i compared to a reference j with:

$$S = \frac{\text{FOM}_i}{\text{FOM}_j} \tag{4}$$

In the following, we will focus on the computation time and the speed-up factor. Regarding the speed-up factor, we compute it only for cells having a relative standard deviation below 20% in both compared methods. The other cells are discarded.

3.3.1 Scores (means and standard errors) update

To illustrate the potential of porting the code to GPU, we first record the average time spent per batch to update

Table 1. *se*TLE algorithmic steps on CPU compared to the TLE average batch in terms of computing time. Comparison done with configuration 1 (Fig. 2, top-left) for $1E+04$ particles par batch as a function of the splitting multiplicity.

Splitting multiplicity NB_{split} for the *se*TLE on CPU	1	4	8	16	32
Relative average time per batch for (Ω, E) sampling (%)	1.57×10^{1}	6.06×10^{1}	1.18×10^{2}	2.38×10^{2}	5.48×10^{2}
Relative average time per batch for cross-sections calculation (%)	8.67	3.48×10^{1}	6.90×10^{1}	1.38×10^{2}	2.64×10^{2}
Relative average time per batch for pseudo-particles transport (%)	1.04×10^{2}	4.07×10^{2}	8.03×10^{2}	1.61×10^{3}	3.14×10^{3}

Table 2. Simulation time comparison for a fixed number of batches with configuration 1 (Fig. 2, top-left). Splitting multiplicity is equal to one, bank of 2^{16} rays, 1024 threads per bloc.

	Scores update on CPU (s)	Scores update on GPU (s)	
Number of cells		Synchronization on	Synchronization off
1.00×10^{7}	3.20×10^{-1}	1.68×10^{-2}	9.90×10^{-6}
1.00×10^{6}	3.20×10^{-2}	1.84×10^{-3}	9.60×10^{-6}
1.00×10^{5}	3.20×10^{-3}	2.94×10^{-4}	8.80×10^{-6}
1.00×10^{4}	3.40×10^{-4}	4.76×10^{-5}	9.50×10^{-6}
1.00×10^{3}	5.60×10^{-5}	2.60×10^{-5}	9.50×10^{-6}

the overall score along the simulation with the new estimated tallies. The results are reported in Table 2. This evaluation is independent of the estimator, whether it is implemented on GPU or not. It is also independent of the sampling on GPU. The average computation time is compared, with and without the synchronization of the CPU and the GPU. When enabled, the synchronization (right after the call to the GPU kernel) forces the CPU to wait for the device to finish the kernel execution. We observe that the time spent to accumulate the scores across the batches linearly increases both on CPU and GPU with the number of cells in the grid. This is verified up to 10^7 cells and cannot be evaluated further on CPU due to limitations in TRIPOLI-4®. The batch collection on GPU with the synchronization is about 20 times faster than on CPU. When synchronization is disabled, we only measure the time spent to call the CUDA kernel, which is independent of the mesh size. This corresponds to the standard operation described in Figure 1: the CPU continues the random walk of the real particles simulated while the GPU performs its calculations.

3.3.2 Ray Tracing on GPU

We are now quantifying the contribution of ray tracing on GPU. To that end, we compare the FOM of the *e*TLE estimator using an implementation of the ray-tracing on CPU and the FOM of the *e*TLE using our implementation of the ray-tracing on GPU (*se*TLE GPU with a splitting multiplicity of 1). This way, a similar number of rays are cast across the geometry. Sampling is done on CPU, and we evaluate the performance of both settings behind the wall in the bunker configuration (Fig. 2, top-left). In this setting, most of the implementations on CPU and GPU are similar. The main differences are thus the calculation of the ray/primitive intersections and the massive, asynchronous parallelization on GPU. We compare both implementations in terms of speed-up factor in configuration 1 to assess the simulations performances. On average, the ratio behind the wall is 5.29. Twelve cells out of 1×10^6 obtain a speed-up factor below one, meaning that the ray tracing on GPU accelerates the simulation almost everywhere. The cells having a speed-up factor of 3 are those close to the source. The highest values of the speed-up factor are then spread out behind the concrete wall and out of the bunker.

3.3.3 Sample on GPU

Pseudo-particles sampling on GPU is optional; we now activate it and illustrate in Figure 3 its influence on the speed-up factor with respect to splitting multiplicity. In this section, we compare to the TLE, which uses no ray casting at all. Scores are still evaluated in the room behind the wall. The results are obtained for a fixed bank size of 2^{16} rays and a single physical CPU core. When sampling on CPU, the performance improvement of the *se*TLE on GPU quickly reaches a plateau. This is due to the time taken by the CPU to perform the sampling that is limiting since the GPU finishes its calculations much faster. When using the GPU sampling, the speed-up factor linearly increases with the number of splits. A plateau is reached at a splitting multiplicity of 60 with a speed-up factor around 65, when increasing the splitting multiplicity does not help covering the geometry any more.

3.3.4 Discussion

Porting different parts of the code to GPU significantly increases the acceleration factor of the *se*TLE for scoring in a Cartesian mesh. Collecting batches on the GPU highlights not only the acceleration achieved by the graphics

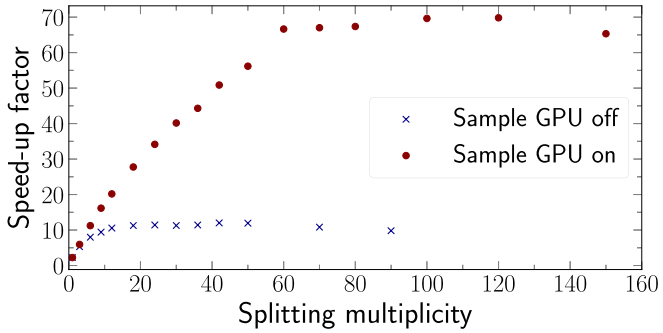

Fig. 3. Influence of the splitting multiplicity on the speed-up factor with and without $(\Omega_{\text{out}}, E_{\text{out}})$ sampling on GPU for the bunker configuration (Fig. 2, top-left).

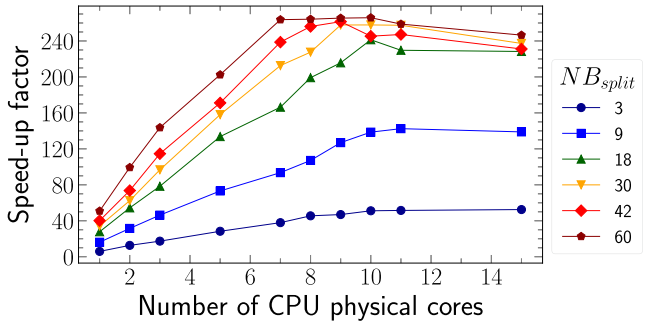

Fig. 4. Parameters study for the bunker configuration: speed-up factor behind the wall to a single CPU TLE function of the number of CPUs used for *se*TLE GPU for different settings.

card, but also the benefits of running CPU and GPU calculations asynchronously. Sampling, ray tracing and batch collection on GPU are carried out asynchronously from the CPU analog random walk of the real particles. This frees up additional CPU computing resources required for *se*TLE. The computation time of the GPU version of the *se*TLE is then at worst the computation time of the CPU version of the TLE. Actually, it will be lower with the batch collection on GPU. Regarding the positive impact of sampling on GPU, it will be used in all the experiments in the rest of the paper. We have therefore detailed the various *se*TLE building blocks that have been implemented on GPUs. The challenge for the GPU is then to be faster than the CPU's computations to make sure that it has finished before the next call. In the workflow of Figure 1, the GPU computations should be completed before the CPU fills the next bank of data to achieve optimal performance. This working point depends on many factors, including the complexity of the functions called, the size of the bank to process, the splitting multiplicity and the average number of collision per real particle. Then one has to adapt the parameters of the *se*TLE GPU to obtain optimal speed-up factors. We chose to leave it up to the user to set them according to their configuration. In the following section, we present the results obtained in the bunker configuration when the parameters are varied.

3.4 Influence of the parameters

We now assess the impact of the different parameters of the *se*TLE on the GPU with the bunker configuration (Fig. 2, top-left): the splitting multiplicity, the number of physical CPU cores used in parallel and communicating with the GPU, and the bank size.

3.4.1 Number of pseudo-particles sampled and number of physical CPU cores

In this section, the score is evaluated in the room behind the wall of the bunker. The results of Figure 3 are now replicated with the use of parallel CPU core random walk computation. We recall that a part of the GPU memory is allocated to each physical CPU core, and that no memory is shared between the cores. This avoids conflict problems, but duplicates memory and GPU calculations. All the results presented are obtained by comparing them with the CPU implementation of the TLE on a single CPU core. The speed-up factor obtained with *se*TLE GPU-based with several CPU cores is illustrated in Figure 4 for different splitting multiplicity. For each splitting multiplicity, two distinct behaviors are observed. Initially, the speed-up factor increases linearly with the number of physical CPU cores involved in the simulation; the higher the splitting multiplicity, the steeper the slope. Subsequently, a plateau is reached between 7 and 11 CPU cores, depending on the chosen multiplicity. In this configuration (configuration 1), the optimal trade-off is achieved with a splitting multiplicity of 60 using 7 physical CPU cores in parallel. The plateaus observed in Figure 4 result from the saturation of the GPU's compute capabilities. It is worth noting that when GPU-based sampling is disabled, this saturation does not occur. In such cases, the GPU consistently processes the particle banks faster than the CPU cores can populate them.

We then evaluate the influence of the splitting multiplicity on the fuel assembly configuration. A single physical CPU core performs the random walk for both the TLE implementation on CPU and the *se*TLE GPU-based implementation. The results are illustrated in Figure 5 and represents the speed-up factor across all mesh cells throughout the entire geometry. As in configuration 1, the speed-up factor increases with the splitting multiplicity. Contrary to the bunker, however, the speed-up factor is averaged over all the cells of the mesh. It reaches a value of 7.54 for a splitting multiplicity of 64, and its spatial repartition is discussed in Section 3.5.2. Interestingly, the speed-up factor plateau is reached before the graphics card is saturated. This means that beyond a certain level of splitting multiplicity, the information provided by the supplementary pseudo-particles no longer compensates for the lower weight associated with each pseudo-particle. This observation also holds Fig. 3 when GPU-based sampling is enabled and the simulation is executed on a single processor. In such cases, the optimal configuration is primarily dictated by the underlying physics and mesh refinement, rather than by hardware limitations. At a splitting multiplicity of 128, the gain in terms of speed-up factor

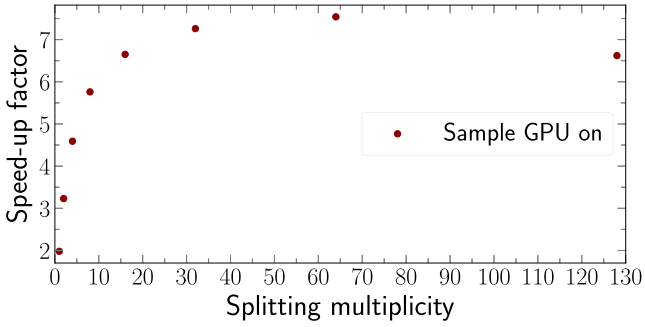

Fig. 5. Speed-up factor in the configuration 2 (the fuel assembly) as a function of the splitting multiplicity. Both TLE CPU and *se*TLE GPU use a single CPU.

decreases: in this case, it is associated with GPU saturation.

3.4.2 Bank size

Similarly to the number of pseudo-particles per collision, the size of the bank sent to the GPU determines the number of operations done on the device at a time. It affects the percentage of the computing capacity used by a single CPU as well as the overall *se*TLE performance. Experiments were carried out for different bank sizes in the bunker configuration (from 2048 to 65538), using five parallel CPUs for the random walk of the real particles, and a splitting multiplicity of 24. Sampling on GPU was enabled, which also increased the computations of the GPU. We observe that the speed-up factor behind the wall increases up to 32768 rays per bank. Over this size of bank, the speed-up factor behind the wall remains constant. This plateau corresponds to an operating point where the bank takes longer to fill than to process. What's more, there is no decrease in the speed-up factor, as the graphics card's computational load is not saturated. The value of the plateau as well as the bank size at which it is reached depend on the splitting multiplicity.

3.4.3 Discussion

Finding a trade-off between the amounts of operations done on GPU for each CPU involved in the simulation depends on many parameters. Some are given by the complexity of the simulation, the number of materials and isotopes involved, the number of volumes in the geometry, or the available hardware. Some others, like the number of volumes in which one has to score, or the parameters specific to the GPU, can be adapted consequently. As illustrated in Figures 4 and 5, the optimal set of parameters may differ drastically depending on the simulation. In the case of the fuel assembly, the number of volumes and the dimensions of the simulation require fewer simulated particles and fewer resources to achieve good performance. In contrast, the bunker configuration is adapted to a higher splitting multiplicity with respect to the small number of materials and isotopes as well as geometric primitives involved.

Fine-tuning the parameters on GPU according to the simulation is a key in optimizing the performances of the GPU implementation of the *se*TLE. To that end, the splitting multiplicity, the size of the bank and the number of threads per bloc can be set in the input data. In any GPU-oriented application, the host should wait as little as possible for the device results to limit the negative impact of the device computations. In this workflow (Fig. 1), the bank should be sufficiently large to guarantee that the device has finished processing the previous bank before the new one is filled. Consequently, the greater the splitting multiplicity, the larger the bank should be.

In addition, we decided to have the CPUs communicate with a single GPU, which linearly increases the device footprint memory and computational load. The aim is to use as many CPUs as possible without saturating the graphics card. One solution we selected for the next section is to use a splitting order of around 24, for 4 or 5 CPUs in parallel and a bank of 2^{16} rays, with the sampling on GPU.

A parameter that was not discussed so far is the number of threads per block involved on the GPU. Its influence can be resumed as follow: when using the maximum number of threads per bloc (1024 for the RTX A4500), the speed-up factor is maximized but it increases the GPU computational burden. One could chose to reduce this number of threads per bloc to enable more CPUs in parallel with one GPU, thus increasing the speed-up factor when comparing to a simulation on one CPU.

3.5 Spatial performances

The performances are now assessed in the three configurations with a focus on the spatial repartition of the speed-up factor obtained with the *se*TLE on GPU when compared to the TLE on CPU. The comparison for both experiments is made using the same number of physical CPU cores. To that end, we boil down to a single physical CPU core for both TLE CPU and *se*TLE GPU. It makes the comparison fairer, since parallel computing on CPUs is now standard in Monte Carlo simulations.

3.5.1 Experiment 1: bunker

In the first experiment (Fig. 2 top-left), the ray tracing on GPU is done using the best set of parameters obtained in Section 3.3: a splitting of 24 and a bank of 2^{16} rays. The number of physical CPU cores used to compute the analog random walks in parallel is five. The speed-up factor per cell is illustrated in a slice of the volume in Figure 6. The first observation is that only 23 cells out of 10^6 have a FOM ratio below 1: all the scores in the other cells are higher, meaning that the *se*TLE on the GPU accelerates the simulation. On average, the speed-up factor is 8.56, and the median is 5.47. Cells with a score below the median are mostly within the room and close to the source. In fact, in the air accessible in a straight line from the source, the TLE CPU and the *se*TLE GPU have similar behaviors since few collisions occur. The contributions to the acceleration factor then come from the rays emitted from the source, and the secondary rays coming from

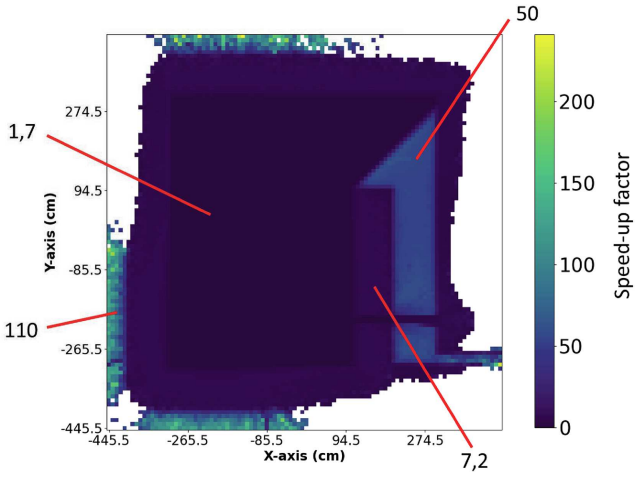

Fig. 6. Acceleration factor of the *se*TLE GPU compared to the TLE CPU in the bunker configuration. White pixels correspond to discarded cells for which the relative standard deviation is above 20% for either *se*TLE GPU or TLE CPU.

the walls. The acceleration factor in the area close to the source is about 1.7. The overall average score is increased by high speed-up values behind the shield and outside the bunker. Note that by taking the number of CPU cores used for the simulation into account, one retrieves the speed-up factor of 250 behind the wall illustrated in Figure 4.

Outside the walls of the bunker, two categories of cells are distinguished. The cells discarded (because of too high a standard deviation) are those for which the segment between them and the source crosses a long portion of concrete; hence, a hard exponential attenuation of the particle weight. The cells for which the FOM is evaluated outside the room have enough statistical information for both TLE CPU and *se*TLE GPU. In that case, the acceleration factor exceeds 100. Note that a majority of the cells outside the bunker were discarded due to a too low MC convergence only for the TLE CPU.

3.5.2 Experiment 2: nuclear fuel assembly

We evaluate the neutron flux in the criticality simulation in the nuclear fuel assembly. A splitting multiplicity of 24 and a bank of 2^{16} are used. The number of physical CPU cores used in parallel is three. This set of parameters provides a good trade-off between the total simulation time and the average speed-up factor over the geometry. The simulation results and the speed-up factor map are illustrated in Figure 7. In the center slice of the assembly, one can observe in Figures 7a and 7b that the tally map generated by the TLE CPU is less converged and noisier than one of the *se*TLE on GPU. On average, the speed-up factor over the cells are 6.30, and the median is 6.67. In total, five cells obtain a speed-up factor less than one. The *se*TLE GPU shows lower performance in geometry boundaries, which may be explained by boundary conditions. The multiplicity order of the *se*TLE for boundary conditions is set to 1, meaning that, in the case of the *e*TLE alone, only the real particle is duplicated and transported along a straight path. In contrast, in the inner part of the assembly, the speed-up factor increases, reaching its maximum of 16.2 in the burnable poison tubes (Fig. 7c).

3.5.3 Experiment 3: dogleg

In order to assess the performance of the *se*TLE method relative to the standard TLE across different geometric regions, the dogleg configuration described in Section 3.1 is particularly well-suited. This is due to the placement of detectors upstream of, within, and downstream of the shielding structure. Table 3 presents the results for the photon source simulation, while Table 4 summarizes the results for the neutron source. The simulation parameters for the photon source simulation using *se*TLE include a splitting multiplicity order of 84 and a bank size of 2^{16}. On average, each photon history undergoes approximately 11 collisions. For the neutron source simulation, the splitting multiplicity order is set to 62, with an increased bank size of 2^{17}. In this case, each neutron history results in an average of 70 collisions. The lower splitting multiplicity and larger bank size for the neutron case are motivated by the significantly higher number of collisions per particle history. Both simulations are performed on a single physical core. The simulation time for TLE is 5.2×10^6 s and for *se*TLE is 6.04×10^5 s for the photon configuration, and 5.39×10^6 s and 5.39×10^5 s respectively for neutrons. For the configuration involving the photon source, the acceleration factor is on the order of unity for detectors located directly in front of the source. It increases approximately 50–65 in the other legs, and reaches values around 100 for the detectors positioned downstream of the shielding. For the configuration with the neutron source, the acceleration factor is approximately 40–55 in the two orthogonal bends, and reaches about 60 in the region located behind the shielding.

3.5.4 Discussion

In each of the three configurations, the GPU-based *se*TLE version brings significant speed-up factors over the detectors of the geometries when compared to the CPU-based TLE. The different configurations cover neutron and photon sources, point sources as well as extended ones. The simulations also address shielding and criticality scenarios, and the detectors consist of meshes mapping the geometry or spherical detectors in the case of the dogleg configurations. This illustrates the contribution of the expectation estimator (which, theoretically, outperforms its associated estimator) when leveraging the GPU's capability. Spatial analysis also enables a better understanding of the *se*TLE GPU's contribution: indeed, the distribution of speed-up factors shows correspondences from one configuration to another. First, in the nuclear fuel assembly, the highest speed-up factors are achieved in the burnable poison tubes (Fig. 7b), where the particles are more likely to be absorbed. This results in fewer intersections between the paths of real particles and the cells for the TLE. Similar observations can be made on the concrete walls of the bunker configuration (Fig. 6). However, the source term in the nuclear fuel assembly covers the entire geometry and thus the highest speed-up factors in the poison tubes. In the bunker and in the dogleg, the point

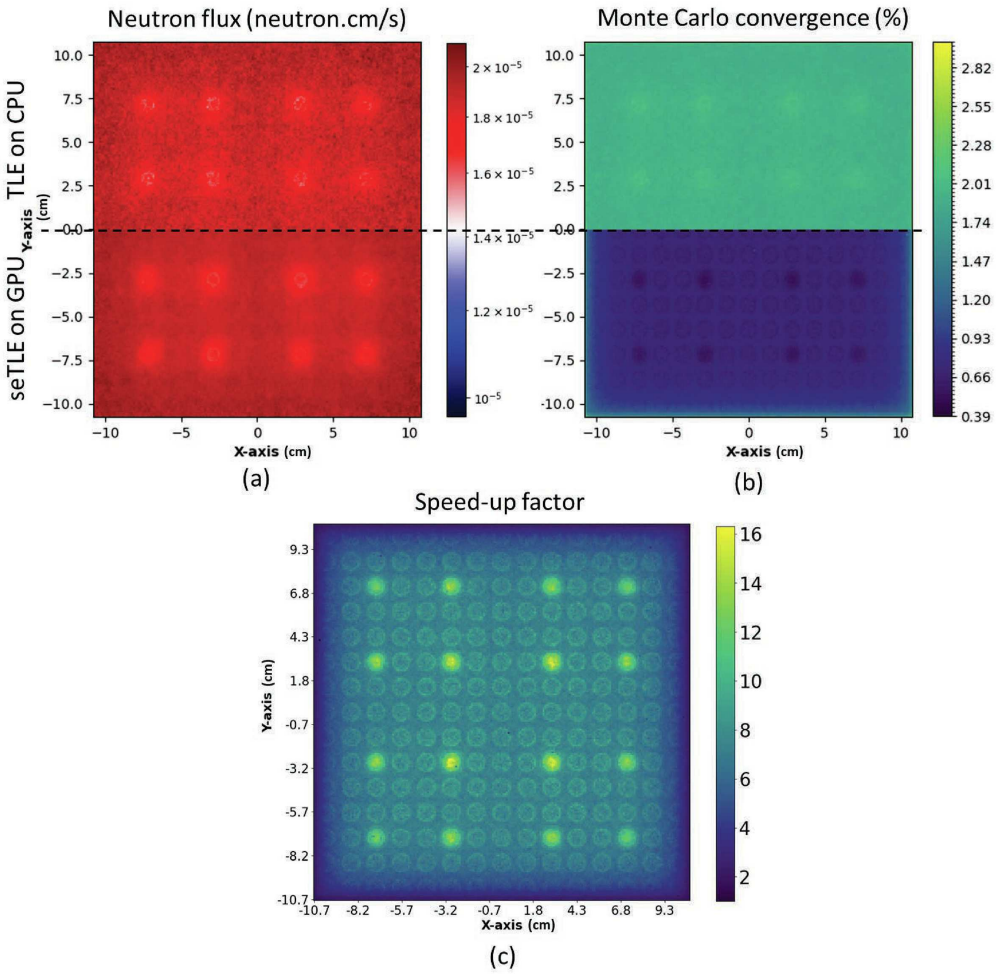

Fig. 7. Neutron flux in a nuclear fuel assembly in neutron.cm/s (a), standard deviation in percent (b) for the TLE on CPU and the *se*TLE on GPU. Speed-up factor of the *se*TLE GPU compared to the TLE CPU (c).

source terms make some regions of the geometries even less likely to be crossed by the paths of the real particles. Such areas, downstream of the shielding, show the highest speed-up factors. In addition to the low statistical information given by the real particles to the TLE, the detectors in these areas are located in the air, thus a higher statistical weight of the pseudo-particles when compared to detectors inside the optically thick materials. In the dogleg configuration, the highest speed-up factors are achieved downstream of the shielding, where the contributions of the real particles are less important whereas the pseudo-particles, mostly emerging from collisions on the walls of the left-hand side room do not cross any obstacle before reaching the detectors. In both the bunker and the dogleg with neutrons or photons, the lowest speed-up factors are achieved within short optical length from the source.

In the bunker simulation, scoring outside the bunker shows little interest in practice, except at the exit where someone could stand. From a development point of view, we observe areas (Fig. 6) where the speed-up factor is discarded because of high standard errors for both estimators. In such areas, neither the CPU-based TLE nor the GPU-based *se*TLE provides information. For the first, the real particles hardly reach these areas while for the second, the attenuation of the concrete walls is too strong, thus a very low statistical weight for the particles are reached. In-between these two extrema, there are areas outside the bunker where the real particles do not go while the pseudo-particles are less attenuated. In that case, the speed-up factor may exceed 100 (e.g., bottom left of Fig. 6).

It must be emphasized that the *se*TLE enables substantial gains across numerous detectors, where conventional VRT-type methods would typically require customized setups for each individual tally. As a result, when scoring in multiple detectors, the *se*TLE offers a more straightforward and user-friendly solution. In certain configurations, the *se*TLE could even be combined with VRT approaches for instance, by constraining the sampling of pseudo-particles within a specific angular domain to preferentially direct rays toward regions of interest. Moreover, one could envision a similar implementation for a wide range of forced-flight techniques, including point detectors, surface-based DXTRAN, or even forced-detection *e*TLE-type methods.

Table 3. Results for the dogleg photon source configuration using TLE and *se*TLE, and associated speed-up factor.

Detector	TLE flux [$cm^{-2}\,s^{-1}$]	TLE std [%]	*se*TLE flux [$cm^{-2}\,s^{-1}$]	*se*TLE std [%]	Δ/σ	Speed-up $\frac{FOM}{FOM_{TLE}}$
1	1.793×10^{-5}	0.07	1.792×10^{-5}	0.17	0.36	1.4
2	8.955×10^{-6}	0.10	8.942×10^{-6}	0.26	0.51	1.2
3	4.657×10^{-6}	0.13	4.641×10^{-6}	0.37	1.12	1.1
4	5.957×10^{-7}	0.37	5.985×10^{-7}	0.15	–1.20	50.0
5	1.015×10^{-7}	0.89	1.012×10^{-7}	0.33	0.33	63.3
6	2.692×10^{-8}	1.73	2.735×10^{-8}	0.61	–0.88	68.7
7	2.791×10^{-9}	5.39	2.740×10^{-9}	1.67	0.33	91.8
8	1.422×10^{-8}	2.43	1.389×10^{-8}	0.70	0.91	104.7
9	1.282×10^{-9}	7.99	1.303×10^{-9}	2.17	–0.20	116.7
10	2.584×10^{-10}	18.36	2.352×10^{-10}	6.37	0.47	71.4
11	2.933×10^{-10}	16.19	3.246×10^{-10}	11.31	–0.52	17.6

Table 4. Results for the dogleg neutron source configuration using TLE and *se*TLE, and associated speed-up factor.

Detector	TLE flux [$cm^{-2}\,s^{-1}$]	TLE std [%]	*se*TLE flux [$cm^{-2}\,s^{-1}$]	*se*TLE std [%]	Δ/σ	Speed-up $\frac{FOM}{FOM_{TLE}}$
1	1.918×10^{-5}	0.09	1.911×10^{-5}	0.19	1.6	2.2
2	9.422×10^{-6}	0.13	9.438×10^{-6}	0.30	–0.5	1.9
3	3.965×10^{-6}	0.20	3.977×10^{-6}	0.53	–0.56	1.5
4	1.296×10^{-6}	0.34	1.288×10^{-6}	0.18	1.60	36.6
5	4.517×10^{-7}	0.58	4.4771×10^{-7}	0.30	1.35	37.5
6	2.095×10^{-7}	0.85	2.090×10^{-7}	0.43	0.27	38.6
7	2.949×10^{-8}	2.27	2.910×10^{-8}	0.98	0.54	53.8
8	4.251×10^{-8}	1.90	4.150×10^{-8}	0.77	1.16	60.3
9	1.160×10^{-8}	3.63	1.182×10^{-8}	1.48	–0.49	59.8
10	4.917×10^{-9}	5.55	5.407×10^{-9}	2.32	–1.63	57.0
11	6.366×10^{-9}	4.90	6.224×10^{-9}	2.13	0.42	53.0

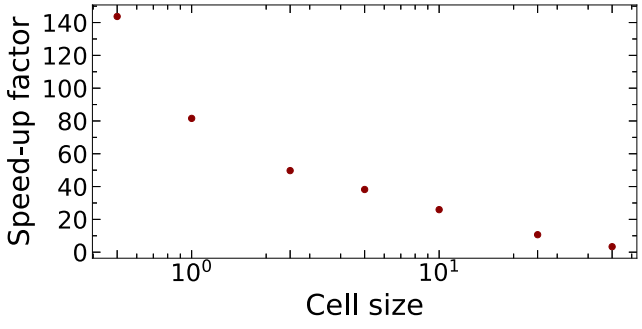

Fig. 8. Average speed-up factor behind the wall between *se*TLE and TLE for different cell sizes in cm.

3.6 Influence of phase space discretization on tallies

One of the key advantages of *se*TLE is its ability to capture contributions across a broader phase space for tallying. Intuitively, as the phase space mesh is more finely discretized for the desired tally, *se*TLE's performance improves relative to the TLE. The divergence in the number of contributions to the statistical quantity between *se*TLE and the TLE becomes more pronounced as the phase space discretization increases. For example, in the case of a bunker, this discrepancy can be spatially observed by varying the cell size behind the shielding. Figure 8 illustrates the speed-up factor behind the wall between *se*TLE and TLE for varying cell sizes, ranging from 0.5 cm to 50 cm for a splitting multiplicity of 42. As expected, the performance of *se*TLE compared to TLE improves as cell size decreases. It is also observed that the trend is not linear, but rather exhibits superlinear behavior. The average speed-up factor is approximately 2 for a cell size of 50 cm, around 38 for 6 cm, and reaches 144 for the finest cell size of 0.5 cm.

This spatial observation also holds for the energy variable, and more generally, for all dimensions of the phase space. In the case of the energy domain, the critical pin cell lattice benchmark provides a relevant example to illustrate this behavior. Figure 9 presents the average speed-up factor per energy group for different energy discretizations, based on the ^{235}U fission rate in the fuel volumes. Four group structures are considered: 50, 200, 800, and 3200 groups. Each finer group structure is nested within the previous one: the 3200-group structure includes the boundaries of the 800-group one, which in turn includes those of the 200-group and the 50-group structures. This nesting ensures a consistent and meaningful refinement comparison. Figure 9 results clearly show that the speed-up factor achieved by the *se*TLE on GPU increases with the resolution of the energy mesh. This trend is consistent with previous spatial analyses involving detectors embedded in poison tubes and concrete walls: the lower the probability of a real particle reaching a given phase-space detector, the higher the potential efficiency gains from the *se*TLE method. Moreover, when multiple phase-space variables are discretized simultaneously, the gain provided by the method corresponds to the product of the individual gains obtained for each variable independently.

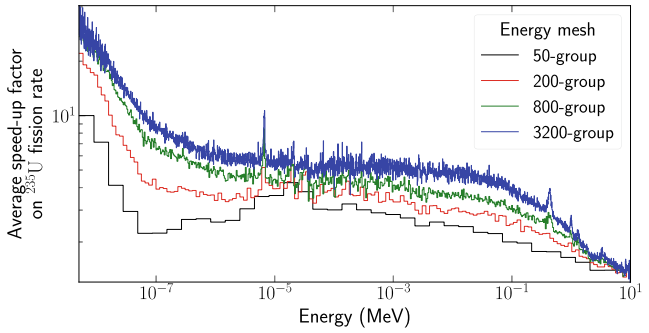

Fig. 9. Average speed-up factor per energy group for different energy discretizations in the fuel volumes of the fuel assembly.

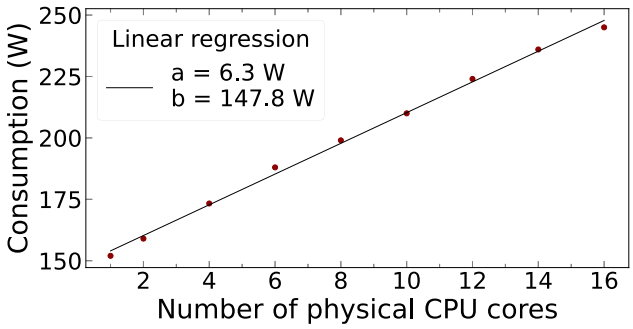

Fig. 10. CPU power consumption used for the simulations.

This observation also suggests that the *se*TLE may not provide a significant acceleration for the evaluation of the effective multiplication factor (k_{eff}) in criticality calculations, compared to the standard TLE. Indeed, this quantity is inherently global, as it involves contributions from all fissile regions of the geometry. However, such an evaluation would incur only a modest computational cost and could be used to complement the combined k estimate with a standalone k_{eff} in order to reduce the overall covariance.

An application where fine spatial resolution is critical for radiography. In this context, the GPU-based *se*TLE could serve as an accelerator-driven imaging technology for neutron and photon radiography. Indeed, a very high splitting multiplicity drastically increases the number of contributions within each pixel. One could even envision forcing each pseudo-particle toward a specific pixel, or a set of pixels, by sampling its direction from a non-analog probability density function.

4 Energetic considerations on FOM and hybrid CPU/GPU implementation

The FOM is a practical and conventional metric in Monte Carlo methods that allows, among other things, the quantification of the gain achieved between a tested method and a reference method, as in this case. This ratio thus provides the user with an estimate of the potential performance achievable using this method, whether on a standard workstation equipped with a sufficiently powerful graphics card or on a heterogeneous supercomputer. This method, well-suited for heterogeneous computing, aligns well with the evolution of scientific computing hardware. However, the FOM ratio does not strictly reflect the actual gain achieved. In fact comparisons involve, on the one hand, a computation performed on a single physical CPU core (TLE) and, on the other hand, a computation executed on a single physical CPU core supplemented by a GPU (*se*TLE). To accurately compare the gain achieved by the hybrid method, the energy consumed is the most rigorously appropriate primary metric.

To this end, the total system power consumption, with and without the GPU, is measured using a wattmeter placed between the wall outlet and the Intel workstation. At idle, the workstation consumes 85 W, rising to 151 W during computation with a single physical CPU core, so $P^{\text{st}}_{\text{core}} = 66\,\text{W}$ consumption associated with the Monte Carlo simulation using a single physical CPU core. As shown in the Figure 10, each additional physical core contributes an average increase of $P^{\text{oth}}_{\text{core}} = 6.3\,\text{W}$ in power consumption. The GPU's contribution to power consumption P_{GPU} is approximated as the difference between the total system power measured by the wattmeter and the power associated with the active physical CPU cores. We illustrate it in the following equation, yet it is part of the wattmeter measurement during the *se*TLE simulations on GPU. In order to assess the energy efficiency improvement introduced by the method, the following formulation is proposed:

$$G_e = S\frac{P_{\text{CPU}}}{P_{\text{CPU}}+P_{\text{GPU}}}, P_{\text{CPU}} = P^{\text{st}}_{\text{core}} + (NB_{\text{core}}-1)P^{\text{oth}}_{\text{core}}$$
$$= \frac{\sigma^2_{\text{TLE}}}{\sigma^2_{se\text{TLE}}}\frac{T_{\text{TLE}}}{T_{se\text{TLE}}}\frac{P_{\text{CPU}}}{P_{\text{CPU}}+P_{\text{GPU}}}$$

where NB_{core} is the number of physical CPU cores used during the hybrid simulation. Ge is the ratio of the energy consumed by each estimator, weighted by the corresponding variance ratio. We illustrate the energy savings for different settings on Figure 11 and Table 5. To produce the first results, we use the same bunker configuration as in Section 3.4. In particular, the Cartesian mesh is $1000 \times 1000 \times 1000$. It is recalled that the FOMs are averaged over the cells located behind the shielding wall. The energy gain decreases with the number of physical CPU cores. This behavior arises because each additional core adds only $\sim 6.3\ W$ of local power consumption, yet produces a similar amount of data as the first core, which must be processed by the GPU. Consequently, the GPU becomes increasingly loaded, leading to a higher rise in its energy consumption. In the expression for Ge, the denominator grows faster than the numerator.

In the case of the dogLeg configuration, photons exhibit a better energy saving factor, primarily due to a higher speed-up factor and because the scattering sampling process is simpler compared to neutrons. As a result, the GPU operates more efficiently, leading to lower power consumption. The GPU power consumption reaches 89 W for the photon case (splitting number 84), versus 91 W for the neutron case (splitting number 62). The optimum

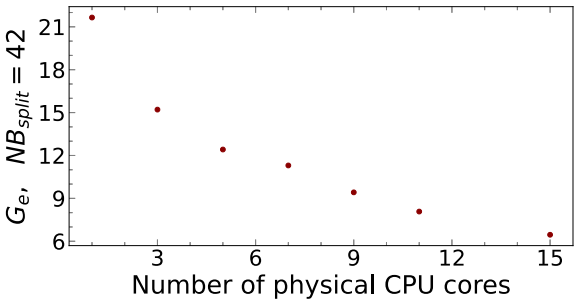

Fig. 11. Energy saving factor for tally behind the wall between *se*TLE and TLE for a splitting multiplicity of 42 as a function of the number of physical CPU cores used.

Table 5. Energy saving factor for the dogLeg configuration with photon and neutron sources.

Detector	G_e source photon	G_e source neutron
1	0.6	0.9
2	0.5	0.8
3	0.5	0.6
4	21.3	15.4
5	27.0	15.7
6	29.3	16.2
7	39.1	22.6
8	44.6	25.3
9	49.7	25.1
10	30.4	24.0
11	7.5	22.3

is reached at a value of 50 for photons and 25 for neutrons, for the best detectors. Overall, these results are less impressive than the acceleration gains in terms of FOM, but they need to be put into perspective. Firstly, we have seen that these gains could be even greater if we used an even finer mesh. Secondly, they can be significantly greater than 1 despite the higher energy consumption of the GPU compared with that of the physical CPU cores. As more and more supercomputers are equipped with graphics cards, this solution saves the user both simulation time and energy consumption. This double saving shows that the solution is perfectly in tune with today's challenges. It also appears that GPU consumption does not evolve linearly (and even less rapidly) with the number of physical CPU cores used in parallel: it remains interesting from an energy point of view to use as many as possible without saturating the graphics card. Finally, it can also be noted that the gains with the method are obtained in many volumes of the phase space. If we were to compare it with the usual VRT methods, which have to be rerun for each volume, we can also expect interesting performances.

5 Conclusion

The effort in this work was to minimize the influence of all parts of *se*TLE that increased the total CPU simulation time by offloading them to the GPU. By doing so, the total simulation time using the GPU sampling and implementation of the *se*TLE becomes shorter than that of the analog random walk of the real particles and the scoring of the TLE on CPU. Thus, porting the *se*TLE to the GPU significantly improves the estimation of volumetric observables such as spatial or energetic flux or reaction rate maps. Recalling the observations in Table 1, we can immediately see that it makes the estimator usable in Monte Carlo simulations.

A study of the influence of the different parameters used in our implementation of the estimator allows a better understanding of the strengths and limitations of the proposed approach. The hybrid CPU-GPU architecture used in the paper enables parallel simulation of real particles from several physical CPU cores while computing *se*TLE on the GPU asynchronously. The limitation is hardware, that is, the speed-up factors obtained with *se*TLE GPU increase linearly with the number of physical cores CPU involved, up to the saturation of the GPU. To achieve the best speed-up factors possible, one has to fine-tune the parameters depending on the simulation to achieve optimal load balancing between CPU and GPU.

We evaluated the performance of our implementation in various experiments, including shielding and criticality, neutron and photon transport, and scoring in spatial Cartesian mesh or spherical volumes and energetic grids. Apart from a few volumetric detectors close to the point sources in the bunker configuration, all the speed-up factors obtained exceeded 1 when comparing the GPU-based *se*TLE and CPU-based TLE. The speed-up factor often reaches a few dozens in areas of interest of the phase space and exceeded 100 in areas hard to access for the real particles. In addition, the last study of Section 4 showed that the proposed implementation allows for an energy saving despite involving a supplementary GPU. That makes it suited to the hybrid CPU-GPU clusters, and is in line with the need for a reasonable energetic consumption.

This implementation paves the way to simultaneous tally map computation on GPU combined with more classic tally estimations in sparse and limited number of detectors on CPU. Such solutions, with adapted variance reduction techniques, could allow for scoring in places where standard TLE or the *se*TLE are limited.

It is important to note that the method can be applied to estimate any quantity that is traditionally evaluated using the standard TLE or collision estimators. Moreover, the straight-line transport of pseudo-particles in asynchronous way on GPU naturally extends to point estimators, DXTRAN forced-flight techniques, and other methods within this family. Applications range from reactor physics and radiation shielding to radiographic imaging.

Currently, one drawback of the implemented ray tracing is the absence of acceleration structure such as the Axis Aligned Bounding Boxes and the Boundary Volumes Hierarchy. Another perspective will be to integrate other geometric primitives than spheres boxes and cylinders to match the complexity of nuclear installations

such as a reactor core. An interesting development would be to implement the ray tracing with a dedicated library, such as Optix [38], developed by NVIDIA, to manage the ray tracing in the geometry in an optimal way.

Acknowledgments

We acknowledge the financial support of the Cross-Disciplinary Program on Numerical Simulation of CEA, the French Alternative Energies and Atomic Energy Commission.

Funding

This research received no external funding.

Conflicts of interest

The authors have nothing to disclose.

Data availability statement

This article has no associated data. Some information can be made available by request to the corresponding author.

Author contribution statement

Conceptualization: H. Hutinet and P.L. Antonsanti. Methodology: H. Hutinet and P.L. Antonsanti. Software: H. Hutinet and P.L. Antonsanti and D. Mancusi. Validation: H. Hutinet and P.L. Antonsanti. Resources: C. Le Loirec. Data Curation: C. Le Loirec and H. Hutinet. Writing – Original Draft Preparation: H. Hutinet and P.L. Antonsanti. Writing – Review & Editing: C. Le Loirec and D. Mancusi. Visualization: D. Mancusi. Supervision: C. Le Loirec. Project Administration: C. Le Loirec. Funding Acquisition: C. Le Loirec.

References

1. I. Lux, L. Koblinger, *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations* (1991). https://doi.org/10.1201/9781351074834
2. J. Spanier, Two pairs of families of estimators for transport problems, SIAM J. Appl. Math. **14**, 702 (1966)
3. G.A. Bird, Direct simulation and the boltzmann equation, Phys. Fluids **13**, 2677 (1970)
4. J.A. Kulesza, T.R. Adams, J.C.E.A. Armstrong, MCNP® Code Version 6.3.0 Theory & User Manual, Tech. Rep. LA-UR-22-30006, Rev. 1 (Los Alamos National Laboratory, Los Alamos, NM, USA, Sept. 2022). https://doi.org/10.2172/1889957
5. S.N. Cramer, Application of the Fictitious Scattering Radiation Transport Model for Deep-Penetration Monte Carlo Calculations, Tech. rep. (Oak Ridge National Lab., TN (USA), 1977)
6. H. Kahn, T. Harris, Estimation of particle transmission by random sampling, in National Bureau of Standards Applied Mathematics Series (1951), Vol. 12, pp. 27–30
7. F.H. Clark, The Exponential Transform as an Importance-Sampling Device. *A REVIEW*, Tech. rep. (1966). https://doi.org/10.2172/4547233
8. T.E. Booth, Genesis of the Weight Window and the Weight Window Generator in MCNP – A Personal History Los Alamos Report LA-UR-06-5807, Tech. rep. (2006). https://doi.org/10.13140/RG.2.1.1254.1608
9. E. Woodcock, Techniques Used in the GEM code for Monte Carlo Neutronics Calculations in Reactors and Other Systems of Complex Geometry, Tech. rep. (Argonne National Laboratory, 1965)
10. J.F. Williamson, Monte Carlo evaluation of kerma at a point for photon transport problems: Monte Carlo calculation of kerma at a point, Med. Phys. **14**, 567 (1987)
11. E.M. Gelbard, L.A. Ondis, J. Spanier, A New Class of Monte Carlo Estimators, SIAM J. Appl. Math. **14**, 697 (1966). https://doi.org/10.1137/0114058
12. T.E. Booth, K.C. Kelley, S.S. McCready, Monte Carlo Variance Reduction Using Nested Dxtran Spheres, Nucl. Technol. **168**, 765 (2009). https://doi.org/10.13182/NT09-A9303
13. M.H. Kalos, On the Estimation of Flux at a Point by Monte Carlo, Nucl. Sci. Eng. **16**, 111 (1963)
14. J.P. Morgan, A. Mote, S.L.E.A. Pasmann, The Monte Carlo Computational Summit – October 25 & 26, 2023 – Notre Dame, Indiana, USA, J. Comput. Theor. Transp. **53**, 361 (2024). https://doi.org/10.1080/23324309.2024.2354401
15. ORNL, *Oak Ridge National Laboratory Frontier*. https://www.olcf.ornl.gov/olcf-resources/compute-systems/frontier/
16. CEA/DAM, *TERA, La saga des supercalculateurs* (2021).
17. F.B. Brown, W.R. Martin, Monte Carlo methods for radiation transport analysis on vector computers, Prog. Nucl. Energy **14**, 269 (1984). https://doi.org/https://doi.org/10.1016/0149-1970(84)90024-6
18. 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). https://doi.org/https://doi.org/10.1016/j.anucene.2014.10.039
19. 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). https://doi.org/https://doi.org/10.1016/j.anucene.2019.03.024
20. S.P. Hamilton, T.M. Evans, Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code, Ann. Nucl. Energy **128**, 236 (2019). https://doi.org/https://doi.org/10.1016/j.anucene.2019.01.012
21. 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). https://doi.org/https://doi.org/10.1016/j.anucene.2021.108508
22. T. Chang, E. Brun, C. Calvin, Portable Monte Carlo Transport Performance Evaluation in the PATMOS Prototype, in *Parallel Processing and Applied Mathematics* (Springer International Publishing, 2020), pp. 528–539
23. A. Klinvex et al., An asynchronous GPU-enabled cross-section lookup algorithm for Monte Carlo transport simulations, in *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M and C 2021)* (2021)
24. E. Brun et al., TRIPOLI-4®, CEA, EDF and AREVA reference Monte Carlo code, Ann. Nucl. Energy **82**, 151 (2015). https://doi.org/https://doi.org/10.1016/j.anucene.2014.07.053
25. F.-X. Hugot et al., Overview of the TRIPOLI-4 Monte Carlo code, version 12, EPJ Nucl. Sci. Technol. **10**, 17 (2024). https://doi.org/10.1051/epjn/2024018
26. E. Guadagni, C. Le Loirec, Y. Pénéliau, A new hybrid next-event estimator for photon-based Monte Carlo dose rate calculations, Eur. Phys. J. Plus **136**, 1135 (2021). https://doi.org/10.1140/epjp/s13360-021-02120-5

27. H. Hutinet, C. Le Loirec, D. Mancusi, Neutron elastic scattering kernel for Monte Carlo next-event estimators in TRIPOLI-4®, Eur. Phys. J. Plus **138**, 189 (2023). https://doi.org/10.1140/epjp/s13360-023-03787-8
28. H. Hutinet, Développement d'un estimateur déterministe appliqué' au transport de particules neutres en méthode Monte-Carlo, Ph.D. thesis, École Doctorale Physique et sciences de la matière (Marseille), 2024
29. F. Smekens et al., Simulation of dose deposition in stereotactic synchrotron radiation therapy: A fast approach combining Monte Carlo and deterministic algorithms, Phys. Med. Biol. **54**, 4671 (2009)
30. J. Sweezy, Photon Next-Event Estimator Implementation in MCATK, Tech. Rep. LA-UR-16-28780 (Los Alamos National Lab, 2016)
31. L. Jahnke et al., GMC: A GPU implementation of a Monte Carlo dose calculation based on Geant4, Phys. Med. Biol. **57**, 1217 (2012)
32. J. Sweezy, A Monte Carlo Volumetric-Ray-Casting Estimator for Global Fluence Tallies on GPUs, J. Comput. Phys. **372**, 426 (2018)
33. NVIDIA, P. Vingelmann, F.H.P. Fitzek, *CUDA, release: 10.2.89* (2020)
34. D. Brusa et al., Fast sampling algorithm for the simulation of photon Compton scattering, Nucl. Instrum. Methods Phys. Res. **379**, 167 (1996)
35. A. Zoia et al., Doppler broadening of neutron elastic scattering kernel in TRIPOLI-4®, Ann. Nucl. Energy **54**, 218 (2013)
36. B. Welford, Note on a method for calculating corrected sums of squares and products, Technometrics **4**, 419 (1962)
37. M. Lei et al., Analysis of Dogleg Duct Experiments with 14-MeV Neutron Source using TRIPOLI-4 Monte Carlo Transport Code, IEEE Trans. Plasma Sci. **46**, 1180 (2018)
38. S.G. Parker et al., Optix: A general purpose ray tracing engine, ACM Trans. Graph. **29**, 1 (2010)

Cite this article as: H. Hutinet, P.-L. Antonsanti, C. Le Loirec, and D. Mancusi. Accelerating split-exponential track length estimator on GPU for Monte Carlo simulations, EPJ Nuclear Sci. Technol. **11**, 62 (2025). https://doi.org/10.1051/epjn/2025053