Proceedings of the Reactor Physics Asia 2023 (RPHA2023) Conference
Gyeongju, Korea, October 24-26, 2023

GPU-optimized Monte Carlo code development - preliminary results

Muhammad Rizwan Ali, Murat Serdar Aygul, Deokjung Lee*
Ulsan National Institute of Science and Technology, UNIST-gil 50, Eonyang-eup, Ulju-gun, Ulsan, 689-798, Korea
**Corresponding author: deokjung@unist.ac.kr*

1. Introduction

Monte Carlo (MC) method is usually employed for high-fidelity particle transport problems where thousands of particles are simulated. MC codes can handle complex geometries and hence provide the most accurate representation of a nuclear reactor among the spectrum of available methods. The expected value of a physical quantity depends on the behavior of the particles that travel within the geometry of the problem. Hence, the computational intensiveness of the MC method to obtain an accurate solution is prohibitively large. Specifically, solving fuel depletion, reactor transients, and multi-physics problems in a reasonable time is still a challenge.

The advancements in, and cost-effectiveness of, graphical processing units (GPUs), have led to a shift of an increasing number of applications to GPU architecture. In this context, the MC method can also achieve much higher floating-point operations per second (FLOPS) with the utilization of GPUs.

MCS, developed at UNIST, is a high-fidelity neutron transport code that is highly optimized for CPU systems and hence shows very good scalability. It can efficiently utilize CPU-cluster systems employing both shared-memory and distributed-memory parallelism [1]. Although a code like MCS can efficiently utilize the available resources, a large computer cluster is required to attain a more realistic simulation time. Not everyone can afford a large computer cluster. Hence, an increasing number of MC codes have either been ported or developed for the GPU architecture. WARP is the first continuous-energy MC code developed specifically for GPUs [2]. GUARDYAN is a time-dependent GPU-based MC code that can perform nuclear reactor transient calculations [3]. Shift [4] and PRAGMA [5] are continuous energy MC codes with PRAGMA developed specifically for GPUs while Shift can execute on CPUs as well as GPUs. All of these codes employ different methods and algorithms to enhance the performance.

In line with the requirements of reducing the simulation time, UNIST has commenced the development of a new GPU-optimized MC code GREAPMC (**G**pu-optimzed **REA**ctor **P**hysics **M**onte **C**arlo) in CUDA C++. To focus on the optimization of geometric and neutron tracking routines, the material treatment in GREAPMC has been simplified in the form of multigroup cross-sections. This paper presents algorithmic choices and modifications with reference to MCS, initial results, and future directions for the further development of the GPU-optimized code GREAPMC.

2. Background

2.1 GPU Architecture

A GPU can execute thousands of threads concurrently, thus providing massive parallelization. In CUDA, 32 threads are clustered together into a *warp*. The threads in a warp execute instructions in a lockstep fashion. A reduced workload on the warp or branching instructions where some threads have nothing to do or some threads take different execution paths reduces the performance of the code. Hence it is always recommended to lessen the number of branching instructions while developing a GPU code. A collection of warps forms a *thread block*. The computational hardware of NVIDIA GPUs is divided into *Streaming Multiprocessors* (SMs). Each thread block is executed on one SM. In an MC GPU code, a particle is assigned to one thread. Consequently, the number of threads can far exceed what the hardware can physically accommodate. A scheduler efficiently manages the loading and unloading of thread blocks onto SMs, adapting based on whether the threads within a thread block have executed their instruction or are awaiting data retrieval from memory.

2.2 Memory Hierarchy

A GPU consists of various types of memory, as shown in Fig. 1 [6]. Each of them is introduced briefly here in the slowest to the fastest order. The largest chunk of available storage is contained in *global* memory. Global memory is visible to all the threads and it has the lowest bandwidth and the highest latency.

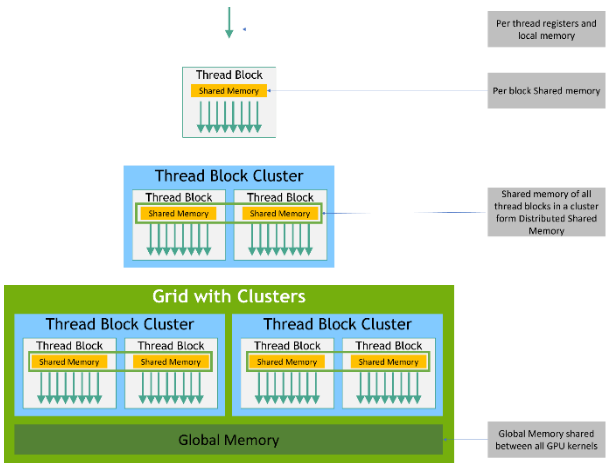
Per thread registers and local memory
Thread Block
Shared Memory
Per block Shared memory
Thread Block Cluster
Thread Block
Shared Memory
Shared memory of all thread blocks in a cluster form Distributed Shared Memory
Grid with Clusters
Thread Block Cluster
Global Memory
Global Memory shared between all GPU kernels

Fig. 1. Memory hierarchy for NVIDIA GPUs.

Therefore, while some threads in a thread block are waiting for data retrieval from global memory, the scheduler loads another warp to execute. This effectively hides the memory latency. Consequently, the number of warps should be large.

The scheduler can make use of a device-wide *L2 cache* to fetch the frequently accessed data. The next level of memory is the *L1 cache* and *shared memory*. These have extremely low latency and are SM-wide (local to thread block) storage but this storage is very limited in size. A GPU also furnishes *thread-private registers* for storing local variables. These are limited in number and when registers are full, variables are spilled to the L1 cache.

The scheduler must choose active warps to execute, considering the available resources like shared memory and registers. As each SM possesses a finite quantity of these resources, an excessive demand from individual threads could lead to a reduction in the active warp count. Consequently, the scheduler may struggle to effectively conceal memory latency.

Hence, high performance on GPUs translates to high occupancy and efficient memory management and data transfer. A GPU card has limited memory and the data transfer from CPU to GPU is via PCIe which can form the bottleneck if the data transfers are not properly managed. Parallelism on GPUs is achieved by Single Instruction Multiple Threads (SIMT) processing. Hence, the slowest thread determines the result from a warp of threads. Thus, it is crucial to distribute the tasks evenly among the threads. An even distribution will increase occupancy and reduce thread divergence.

3. Performance Optimizations

3.1 Neutron Tracking

There are two methods for tracking neutrons within the problem geometry. History-based neutron tracking has been conventionally used for CPUs but for GPUs, event-based tracking has been suggested. The conventional history-based neutron tracking can be detrimental to the GPU performance owing to the variation in history lengths as shown in Fig. 2.

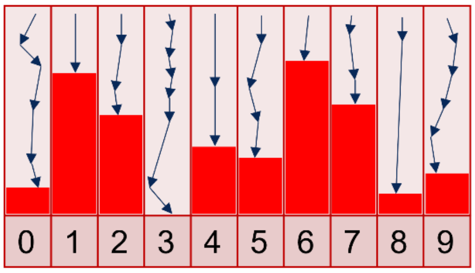

Fig. 2. Conventional history-based neutron tracking depicted for 10 neutrons. The red areas show a waste of GPU resources.

To enhance the performance of the history-based neutron tracking approach on GPUs, Profugus [7] proposed a modified history-based tracking wherein the history length is truncated to a specific number of collisions. Afterward, the neutron population is sorted based on the particle's dead and alive status, and CUDA function (*kernel*) to run on GPU is called with only alive neutrons again. This continues until all the neutrons are dead. The same modified history-based tracking approach was adopted in PRAGMA. The only change was that instead of counting only collision as an interaction, surface crossing was also considered as an interaction [5]. In GREAPMC, the conventional history-based neutron tracking has been modified in the same fashion as in PRAGMA.

3.2 Geometry Treatment

MCS employs Constructive Solid Geometry (CSG) for modeling the problem. Since GREAPMC initially targets PWRs, geometry modeling has been streamlined to encompass square lattices and cylindrical rods. In contrast to surface-oriented geometry construction, this cell-based geometry modeling reduces the complexity, thereby reducing the execution time of geometric functions.

3.3 Particle Data

Maximizing the memory throughput can significantly influence the code's overall performance. Therefore, ensuring memory coalescing is a vital consideration. *Coalescing* is directly related to the arrangement of data in contiguous memory locations. For example, consider a scenario where threads within a warp need to access the particle positions. If the position data is stored in consecutive memory addresses, memory throughput improves. Conversely, when memory accesses follow a strided pattern, the memory throughput diminishes.

The particle data is most frequently accessed during the execution of the transport loop. Therefore, unlike MCS, the particle data in GREAPMC is arranged in a Structure of Array (SOA) pattern. As a result, all the data related to the position of particles is placed contiguously, and so on for other data items.

3. Results and Discussion

This section verifies GREAPMC against MCS using a two-dimensional pin-cell model from C5G7 [8] with reflective boundary conditions. Both MCS and GREAPMC are run with the same multi-group cross-sections. The effective multiplication, volume, and energy averaged flux tally, and fission reaction rate for the pin cell are verified against MCS. Afterward, the single-node performance of GREAPMC over MCS for the two-dimensional C5G7 core problem is presented. The specifications of the CPU and GPU nodes are given in Table I. The GPU node consists of 2 GPU cards, but only one card was used in the current work. Similarly, the CPU contains 64 cores but here the problem was run using a single core. Hence, the comparison is one CPU core against one GPU card. The comparison is fair, as the calculation of effective CPU cores takes into account the number of CPU cores used for the comparison as specified in Eq. (1) [7].

$$\text{Effective CPU Cores} = \frac{(\text{Number of CPU Cores})\times(\text{CPU run time})}{\text{GPU run time}} \quad (1)$$

3.4 Pin-cell Model

Pin-cell model consists of two regions, a moderator and a fuel-clad mixture as shown in Fig. 3.

Table I: Node Specifications

	Quantity	Value
GPU	Model	NVIDIA GeForce RTX 3090
	Base Clock	1695 MHz
	Memory	23.7 GB
CPU	Model	AMD EPYC 7452
	Base Clock	2350 MHz
	Memory	251 GB

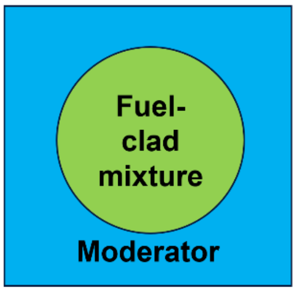

Fig. 3. C5G7 pin-cell model consisting of two regions.

The results of the multiplication factor, reaction rates, and flux tally are presented in Table II. Here 50 inactive, 250 active cycles with 5×10^6 histories per cycle.

Table II: Verification of GREAPMC

Quantity	GREAPMC	MCS
k_{eff}	$1.32562 \pm 1.628\times10^{-5}$	$1.32563 \pm 1.701\times10^{-5}$
Flux Tally	$41.7672 \pm 1.11\times10^{-5}$	$41.7467 \pm 1.068\times10^{-5}$
Fission Reaction Rate	$0.539789 \pm 1.31\times10^{-5}$	$0.539910 \pm 1.195\times10^{-5}$

The results show that the global parameters calculated by GREAPMC are in close agreement with MCS. The reactivity difference comes out to be 0.56 ± 1.33 pcm. Similarly, the absolute difference in the flux tally result is $0.0205 \pm 1.55\times10^{-5}$, while in the fission reaction rate, the absolute difference is $1.21\times10^{-4} \pm 1.77\times10^{-5}$. The standard deviation in flux is smaller than in fission reaction rate as flux is tallied more often compared to fission reaction rate. Currently, efforts are directed toward the implementation of the ability to tally pin-by-pin flux and power within GREAPMC. Fig. 4 shows the core used here for computing the speedup of GREAPMC. The figure shows only xy-view. The boundary condition in the z-direction is also reflective. The core consists of two types of assemblies in line with the C5G7 benchmark. But here full core has been modeled instead of a quarter core.

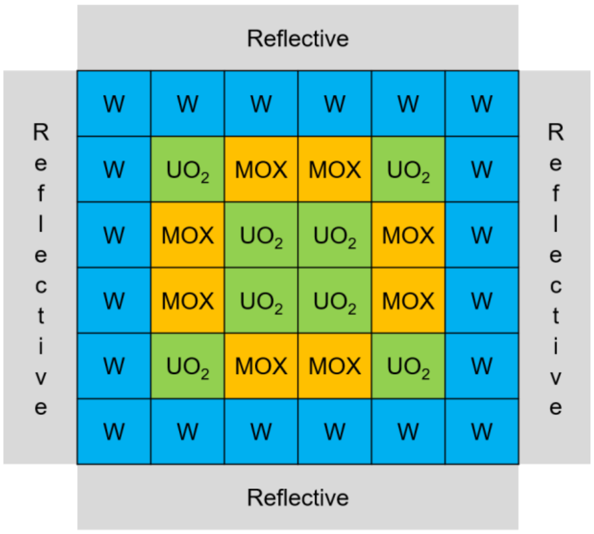

Fig. 4. Core model with reflective boundary conditions

Although the results are computed for multiple numbers of histories per cycle, only the results for 5×10^6 histories per cycle are presented here. The number of active and inactive cycles is the same as in the pin cell case. Table III shows the results. For easy comparison, the time has been normalized against the time that GREAPMC takes to run the problem. Hence, if GREAPMC takes 1 s to run the problem then MCS will take 369 s. The speedup directly computed from this normalization comes out to be 369.

Table III: Comparison of k_{eff} and normalized time

Quantity	GREAPMC	MCS
k_{eff}	$1.18651 \pm 2.01\times10^{-5}$	$1.18650 \pm 2.00\times10^{-5}$
Average time (per cycle)	1 s	369 s

The reactivity difference for this case between GREAPMC and MCS comes out to be 0.71 ± 2.01 pcm. The speedup with the variation in the number of total histories for the core problem is shown in Fig. 5. With increasing the number of histories, the GPU shows an increase in speedup. This shows that with an increase in load, the GPU more effectively hides the memory latency. The speedup seems to saturate as the total number of histories increases. As the speedup directly reflects the tracking rate. Therefore, the tracking rate also saturates with an increase in the number of histories.

3. Conclusions

In conclusion, the ongoing development of the GPU-optimized Monte Carlo code, GREAPMC, at UNIST holds immense promise in reactor simulations. The preliminary performance evaluations show a remarkable speedup compared to MCS. The target goals for the GREAPMC code development are pin-by-pin tally capability, optimization of the neutron tracking approach, continuous energy treatment, and cycle depletion capability.

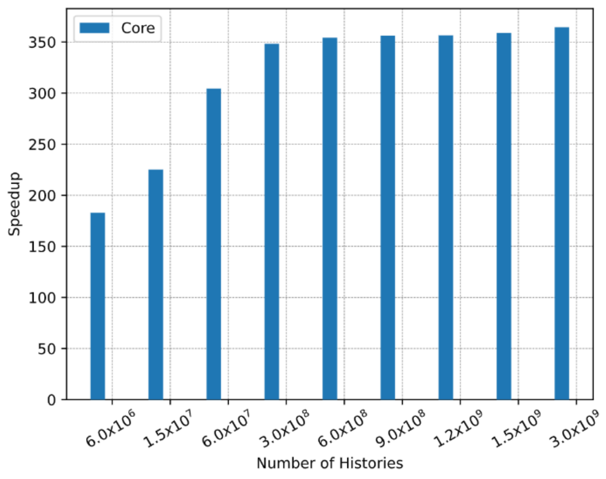

Fig. 5. Speedup plot for the core model

The inclusion of continuous energy treatment ensures greater accuracy in capturing complex reaction behavior. Furthermore, the inclusion of cycle depletion capability will elevate the versatility of GREAPMC, enabling the investigation of long-term reactor behavior.

ACKNOWLEDGEMENTS

This research has been supported by project L20S089000, funded by Korea Hydro & Nuclear Power Co. Ltd.

REFERENCES

[1] H. Lee *et al.*, "MCS – A Monte Carlo particle transport code for large-scale power reactor analysis," *Annals of Nuclear Energy*, vol. 139, p. 107276, 2020/05/01/ 2020, doi: https://doi.org/10.1016/j.anucene.2019.107276.

[2] R. M. Bergmann and J. L. Vujić, "Algorithmic choices in WARP – A framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs," *Annals of Nuclear Energy*, vol. 77, pp. 176-193, 2015/03/01/ 2015, doi: https://doi.org/10.1016/j.anucene.2014.10.039.

[3] B. Molnar, G. Tolnai, and D. Legrady, "A GPU-based direct Monte Carlo simulation of time dependence in nuclear reactors," *Annals of Nuclear Energy*, vol. 132, pp. 46-63, 2019/10/01/ 2019, doi: https://doi.org/10.1016/j.anucene.2019.03.024.

[4] S. P. Hamilton and T. M. Evans, "Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code," *Annals of Nuclear Energy*, vol. 128, pp. 236-247, 2019/06/01/ 2019, doi: https://doi.org/10.1016/j.anucene.2019.01.012.

[5] N. Choi, K. M. Kim, and H. G. Joo, "Optimization of neutron tracking algorithms for GPU-based continuous energy Monte Carlo calculation," *Annals of Nuclear Energy*, vol. 162, p. 108508, 2021/11/01/ 2021, doi: https://doi.org/10.1016/j.anucene.2021.108508.

[6] NVIDIA. "CUDA C++ programming guide." https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#memory-hierarchy (accessed 31 July, 2023).

[7] S. P. Hamilton, S. R. Slattery, and T. M. Evans, "Multigroup Monte Carlo on GPUs: Comparison of history- and event-based algorithms," *Annals of Nuclear Energy*, vol. 113, pp. 506-518, 2018/03/01/ 2018, doi: https://doi.org/10.1016/j.anucene.2017.11.032.

[8] M. A. Smith, L. E. E.;, and N. B-C;, "Benchmark on deterministic transport calculations without spatial homogenisation: A 2-D/3-D MOX fuel assembly benchmark," in "NEA/NSC/DOC(2003)16," 2003.