Initial Development of PRAGMA – A GPU-Based Continuous Energy Monte Carlo Code for Practical Applications

Namjae Choi, Kyung Min Kim, Han Gyu Joo*
Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea
**Corresponding author: joohan@snu.ac.kr*

1. Introduction

The Monte Carlo (MC) method in computational reactor physics provides the significant advantage of simulating neutrons in continuous phase space involving energy, angle and space. It can also use the raw nuclear reaction data directly. However, due to its probabilistic nature, a significant amount of stochastic samples are required to produce statistically reliable results. The excessive computational burden associated with using large samples prevents the use of the MC method in practical reactor design analyses.

The most straightforward approach to alleviate the excessive computing time of the MC method is massive parallelization exploiting the inherent characteristic of mutual independence in particle simulation. The Shift code [1] of the Oak Ridge National Laboratory (ORNL) and the OpenMC code [2] of MIT had demonstrated excellent scaling efficiency with more than 100,000 CPU cores and thus proved the time-wise practical applicability of the MC method on the leadership-class supercomputers.

However, relying on supercomputers is not practical in real sense since such systems are not affordable in most institutions. There can be, however, an alternative of achieving high computing performance using GPUs (Graphics Processing Units) which have become more popular with the rise of artificial intelligence (AI). A pioneering MC neutron transport code, WARP [3] developed at U.C. Berkeley succeeded in showing the potential of continuous energy MC calculation on GPUs.

Motivated by the pioneering research, we recently initiated the development of a GPU-based continuous energy Monte Carlo code PRAGMA (Power Reactor Analysis using GPU-based Monte-Carlo Algorithm) for 'pragmatic' applications. Our experiences in the GPU acceleration of nTRACER [4] became the basis of the development. This paper introduces the challenges in offloading the MC method onto a GPU-based platform. The algorithms and features of PRAGMA, and some preliminary results are presented.

2. Backgrounds and Algorithms

2.1 Characteristics of GPUs and Challenges

GPU is a vector processor and executes instructions in the SIMD (Single Instruction Multiple Data) fashion. A single GPU card contains thousands of cores but they are only partially independent. A core can execute only the instructions independently; fetch or decode is done totally for a group of cores. Therefore, a single dispatch unit controls several cores simultaneously, which means that the cores that belong to same dispatch unit always receive the same instruction, but operate on different data.

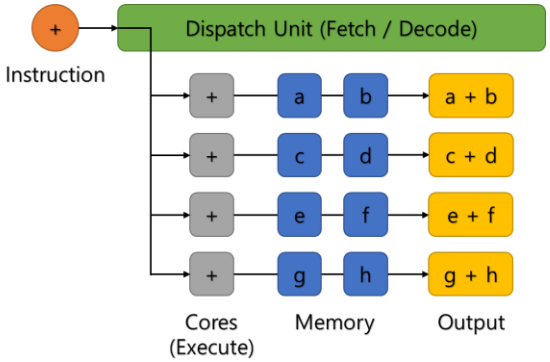

Figure 1. Schematic diagram of SIMD processing.

In CUDA, 32 work items are grouped as a *warp* and processed together. When there are not enough work items that can be processed in SIMD, the remaining threads in a warp are filled with dummy instructions. It is called *thread divergence* and it decreases efficiency.

GPUs typically have large memory bus to provide high memory bandwidth. The memory bus width which is the amount of data transferred at each memory access ranges from 256 to 1024-bit for modern GPUs. As an example, if only a single 32-bit variable is used at each memory access with a 256-bit bus, the achieved memory throughput is only 12.5%. Thus, in order to maximize the memory throughput, adjacent threads should read memory contiguously so that multiple access requests can be handled by a single transaction through the bus. This is called memory *coalescing*.

It should be noted that accessing the main memory is often a bottleneck and each main memory access can waste several hundred compute clock cycles. Although the main memory of GPUs is much faster than that of CPUs, the relative operating speed is slower because GPUs have significantly higher FLOPS. Thus, GPUs contain a large pool of registers, which are fast buffer memories used to save thread-private local variables. Even though the registers are full, the variables are *spilled* to L1 caches which are still fast. Therefore, *pre-fetching* reused data into local variables is crucial to minimize the memory bottleneck.

As the result, GPUs are poor at branching instructions and random memory accesses due to the SIMD and the

coalescing requirement. However, these (branching and random access) are exactly what the MC simulation requires. Such difficulties led to the introduction of a fundamentally different algorithm referred to as *event-based tracking*, in contrast to conventional *history-based tracking* in the MC particle simulation.

2.2 Tracking Algorithm: History-based vs Event-based

The history-based tracking algorithm traces neutrons from their birth to death one by one. It is the most straightforward algorithm, but it has been considered inappropriate for vector processors like GPUs due to the conditional branches. Every neutron undergoes its own sequence of 'events' which is different from others and the choice of an event is randomly determined through conditional branches.

Conversely, in the event-based tracking algorithm which was first introduced in 1980s [5] when vector processing first became popular in the contemporary supercomputing arena, the neutron events are processed one at a time, and a collection of neutrons that are about to undergo the event are tracked together. By processing the event with a SIMD instruction for those neutrons, the vectorization capability of GPUs can be exploited. The efficiency of the event-based tracking algorithm is determined by how well the events are defined and how efficiently sorting is done.

In PRAGMA, a hybrid approach mainly based on the history-based algorithm is introduced. This approach is based on our observations that the most critical process is the cross section reconstruction part, in which random memory accesses are dominant over branches. The event-based algorithm is for reducing branches not the random accesses, so the efficiency of the event-based algorithm is limited. The process where the algorithmic branch is dominant is the post-collision process where fission neutrons are created, reactions are determined, and outgoing energy-angle pairs are sampled. However, the time portion of the post-collision process appeared to be less than ~15%, whose improvement by sorting might be marginal and even outweighed by the sorting cost itself.

In addition, even though the history-based algorithm suffers from event branches, it has a benefit that local memories can be exploited. In the GPU implementation of the history-based algorithm, the entire simulation loop is wrapped into a one large kernel so that the data shared throughout the loop such as the neutron weight, energy, local cross sections, or other indices can be saved in local variables. However, in the event-based algorithm where the event kernels are split, those data should be saved to and read from main memory every time, since local variables cannot exist across kernels.

As the result, the tracking algorithm in PRAGMA is the history-based algorithm with minimal sorting on the status of neutron life and death. As simulation proceeds, neutrons die out and neutron array becomes 'porous.' Such state of active and inactive neutrons being mixed up over the threads causes thread divergences. Hence, at every kernel call, the neutrons are simulated only by a fixed number of iterations and sorted by the life and death status. Then, subsequent kernel begins with only alive neutrons, and finally, the outer loop continues until all the neutrons are dead. The following pseudocode illustrates the hybrid tracking algorithm.

```
while (num_alive > 0)
  Launch GPU kernel with num_alive threads
    parallel foreach num_alive neutrons
      for i = 1 : max_iteration (= 10)
        if (!alive) break
        Calculate macroscopic cross section
        Calculate DTS and DTC
        Move neutron to next position
        if (DTS < DTC) continue
        else
          Sample target nuclide
          Generate fission neutrons
          Sample collision reaction
          Sample outgoing energy-angle pair
          Update weight and perform Russian roulette
        end if
      end for
    end parallel foreach
  Sort alive/dead neutrons using flagged partition
  Update num_alive
end while
```

Figure 2. Hybrid tracking algorithm.

2.3 Cross Section Treatment

In HFP or burned conditions, it might not be possible to store all the point-wise cross sections on a GPU due to the limited memory. Therefore, windowed multipole (WMP) method [6] is employed to reduce the storage required to save cross sections. The Faddeeva function that appear in the formulation is tabulated using wofz() function in SciPy math library and linearly interpolated during calculation. However, a few exceptional nuclides require explicit Faddeeva function calculation because of large residues. For those nuclides, CUDA version of CERNLib Faddeeva function [7] is used.

The cross sections in unresolved resonance energy range and the collision physics data such as fission yield or outgoing energy-angle distribution are not covered by the WMP method and thus read from ACE files. Since the unresolved resonance energy ranges are mostly high enough to neglect the thermal motion of target nuclei, the temperature dependence of cross sections does not have to be taken into account.

In addition, unionized grid and double indexing [8] method is employed for point-wise cross sections. Since the unionized grid is generated only for the energies in

unresolved resonance ranges, the major drawback of the unionized grid method in terms of memory requirement is largely alleviated.

2.4 Optimization for Power Reactor Geometry

The main application targets of PRAGMA are PWRs that have typical lattice geometries. Thus, the geometry module of PRAGMA is optimized for square lattices. The input system and the internal geometry structure follow those of nTRACER so that the same geometry input file can be used.

Limiting the geometrical capability to lattice structure enables cell-based geometry construction. Even though it lacks generality, it is more efficient than surface-based geometry representation where cells are defined as the enclosure of surfaces. In the former, each cell can save its neighbors' indices so that the cell index can be found at no time. However, in the latter, significant overhead is introduced for the cell index search, especially when many cells are introduced in the active cycles to obtain detailed tallies.

There exist millions of cells in a 3D full core model. However, most cells share the same shape, and usually there are only tens of different cell types in terms of shapes. Thus, we define cells with common shapes so that we can call base cells, and only the shapes of the base cells are saved on GPU. The global cells hold their base cell indices and origins of local coordinates.

2.5 T/H Feedback

Currently, a simple 1D closed-channel single-phase model is employed. The fuel temperatures are calculated using the finite difference formulation, and the coolant temperature calculation uses the marching scheme with enthalpy conservation. To compensate the lack of lateral mixing, the coolant temperatures are updated assembly-wise, while the fuel temperatures are updated pin-wise.

3. Results and Discussion

In this section, accuracy and computing performance of PRAGMA are presented. nTRACER input model of APR1400 initial core [9] was directly used. Various size of problems ranging from a single pin to 3D full-core were solved. As the reference code, McCARD [10] was used. The isotope composition of the APR1400 model are listed in **Table 1**.

Table 1. Problem specification.

Region	# of Constituent Isotopes
Fuel	5 or 12 (U, O, Gd)
Gas Gap	2 (N, O)
Cladding	11 (Fe, Zr, Nb, O)
Structure	15 (Cr, Fe, Ni, Mn, Co)
Moderator	4 (H, O, B)

3.1 Verification of Accuracy

The verification was performed for single pins and assemblies of APR1400. Only point-wise cross sections were used because there is no corresponding option in McCARD for the WMP method. The ENDF-B-VII.1 library were used in both codes.

It turned out that the *k*-effective of PRAGMA agrees with that of McCARD for all the pin and assembly types of APR1400 within 10 pcm. **Figure 3** illustrates the flux spectra of the two codes for the 3.65% fuel pin as the representative, which match each other very well. This confirms the sound implementation of the continuous energy calculation kernels in PRAGMA.

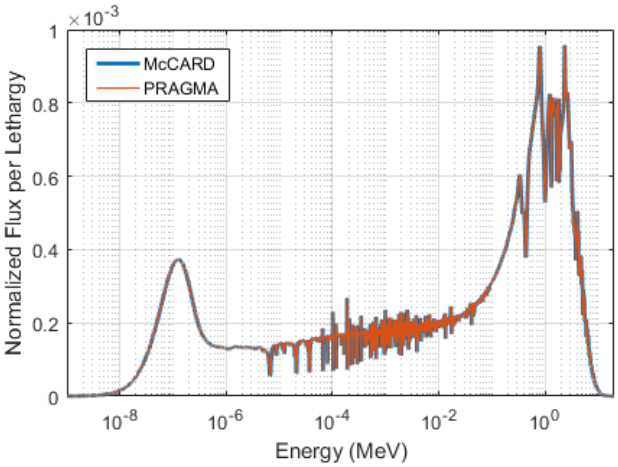

Figure 3. Comparison of single pin flux spectra.

3.2 Performance Analysis

For performance assessment, the following in-house GPU cluster was used. It is a moderate-sized cluster and equipped with cheap consumer-grade GPUs. We expect that the clusters with such specification would be readily affordable in academia and industries.

Table 2. Computing cluster specification.

# of Nodes	6
CPU / Node	2 × Intel Xeon E5-2630 v4
GPU / Node	4 × NVIDIA GeForce GTX 1080
Memory / Node	8 × 16GB DDR4 RAM
Interconnect	Mellanox Infiniband (56Gbps)

First, a 2D quarter core problem was solved with both PRAGMA and McCARD and the computing times are compared in **Table 3**. McCARD used 432 Xeon E5-2640 v3 CPU cores. 24 million histories were tracked per cycle and 200 inactive cycles and 300 active cycles were used.

Both codes tally pin and assembly power during the active cycles, and as can be seen, there is almost no overhead during the active cycles in PRAGMA. Also, PRAGMA achieves significant speedup over McCARD, reaching the performance equivalent to McCARD using

~6,300 cores. This proves the effectiveness of the GPU acceleration algorithms and the dedicated optimizations.

Table 3. Computing time comparison.

Code	McCARD	PRAGMA	Speedup
Inactive (per Cycle)	2:06:04 (37.82s)	0:16:33 (4.96s)	7.6
Active (per Cycle)	8:09:27 (97.89s)	0:25:38 (5.13s)	19.1
Total	10:15:31	0:42:11	14.6

Second, a 3D full-core model with 36 axial nodes was solved using different options. In HFP calculation using ACE library, six temperatures (550, 600, 900, 1200, 1500, and 1800K) were used and linearly interpolated with respect to $T^{2/3}$. 100 million histories per cycle, 20 inactive cycles paired with pin-wise CMFD, and 300 active cycles were used. The computing times and the uncertainties are summarized in **Table 4** and **Figure 4**. Note that the temperature distribution inside the fuel is averaged for neutron transport, so there is no overhead in HFP calculations compared to HZP calculations.

WMP turns out to be much slower than using only ACE libraries due to the relatively high computational cost and the requirement of double precision. Note that for all the other calculations, single precision arithmetic is enough. In burned conditions, the use of WMP will be indispensable but its overhead will grow significantly, so more optimization is required.

The uncertainty of pin and unit cell (axially divided sub-volume inside a pin) powers are sufficiently small. Note that only track-length estimator is used for power tally for the time being.

Table 4. Summary of computing time and uncertainty.

Option	HZP	HFP	
	ACE	ACE	WMP
Time	2:12:01	2:18:18	4:17:10
Pin Power Mean / Max σ	0.09% 0.17%	0.09% 0.16%	0.09% 0.16%
Cell Power Mean / Max σ	0.45% 3.88%	0.45% 3.38%	0.45% 3.42%

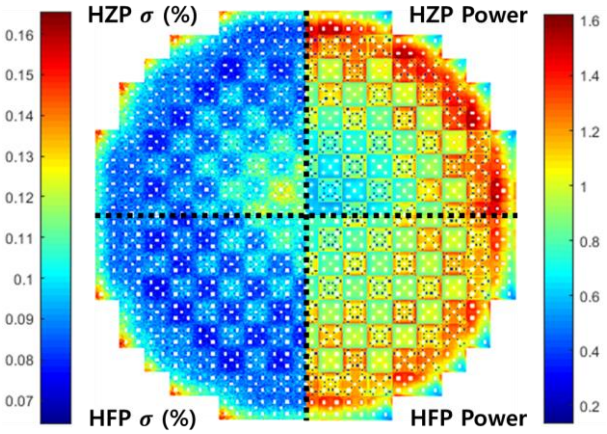

Figure 4. Pin power and apparent standard deviation.

4. Conclusion and Future Work

PRAGMA, a GPU-based continuous energy Monte Carlo code dedicated to analyzing commercial PWRs, is being developed and the preliminary performances are promising. Considerable speedup was achieved against conventional CPU-based MC codes, which proved the effectiveness of the GPU acceleration algorithms and the dedicated optimization strategies. In addition, 3D full-core HFP solutions could be obtained within a few hours using only 24 cheap gaming GPUs. Using gaming GPUs instead of scientific-purpose GPUs like NVIDIA Tesla cards can significantly reduce the cost.

As the future work, further optimizations to improve the performance, especially focusing on WMP, will be carried out. In addition, generalized geometry treatment capability using the NVIDIA ray tracing engine OptiX and cycle depletion capability will be implemented.

ACKNLOGEMENTS

This work was supported by KOREA HYDRO & NUCLEAR POWER CO., LTD (No. 2018-Tech-09).

REFERENCES

[1] T. Pandya *et al.*, "Implementation, Capabilities, and Benchmarking of Shift, a Massively Parallel Monte Carlo Radiation Transport Code," Journal of Computational Physics, 308, pp. 239-272 (2016).
[2] P. Romano and B. Forget, "The OpenMC Monte Carlo Particle Transport Code," Annals of Nuclear Energy, 51, pp. 274-281 (2013).
[3] R. Bergmann, "The Development of WARP – A Framework for Continuous Energy Monte Carlo Transport in General 3D Geometries on GPUs," Ph.D. Dissertation, University of California, Berkeley (2014).
[4] N. Choi, J. Kang, H. G. Joo, "Preliminary Performance Assessment of GPU Acceleration Module in nTRACER," Transactions of the Korean Nuclear Society Autumn Meeting, Yeosu, Korea, Oct. 25-26 (2018).
[5] F. Brown and W. Martin, "Monte Carlo Methods for Radiation Transport Analysis on Vector Computers," Progress in Nuclear Energy, 14(3), pp. 269-299 (1984).
[6] C. Josey, P. Ducru, B. Forget, and K. Smith. "Windowed Multipole for Cross Section Doppler Broadening," Journal of Computational Physics, 307, pp. 715-727 (2016).
[7] A. Oeftiger *et al.*, "Review of CPU and GPU Faddeeva Implementations," Proceedings of the International Particle Accelerator Conference (IPAC), Busan, Korea (2016).
[8] J. Leppänen, "Two Practical Methods for Unionized Energy Grid Construction in Continuous-Energy Monte Carlo Neutron Transport Calculation," Annals of Nuclear Energy, 36, pp. 878-885 (2009).
[9] H. Hong and H. G. Joo, "Analysis of the APR1400 PWR Initial Core with the nTRACER Direct Whole Core Calculation Code and the McCARD Monte Carlo Code," Transactions of the Korean Nuclear Society Spring Meeting, Jeju, Korea, May 18-19 (2017).
[10] H. J. Shim *et al.*, "McCARD: Monte Carlo Code for Advanced Reactor Design and Analysis," Nuclear Engineering and Technology, 44(2), pp. 161-176 (2012).