VECTORIZED MONTE CARLO FOR GUARDYAN - A GPU ACCELERATED REACTOR DYNAMICS CODE

Balazs Molnar, Gabor Tolnai, David Legrady, Mate Szieberth
Institute of Nuclear Techniques, Budapest University of Technology and Economics
3. Muegyetem rkp., Budapest

molnar.balazs@reak.bme.hu, legrady@reak.bme.hu

ABSTRACT

Vectorization of a computer code offers significant speedup of execution time on parallel computing architectures. Vectorized Monte Carlo (MC) simulations require major changes to a conventional algorithm, which generally follows a history-based structure. The non-trivial task of implementation has already been addressed at the time of the first appearance of vector computers, the new simulation structure was termed ”event-based”. With the recent exponential improvement of multicore systems and graphics processing units (GPUs), new challenges and possibilities emerged to optimize parallel algorithms. The efficiency of the event-based algorithm is still disputed. In this paper, vectorization of a GPU accelerated dynamic Monte Carlo code, GUARDYAN, is discussed. Particularly, the performance gain over the history-based version was studied. In a homogeneous test problem speedups of 1.5-2 were typically observed in 2060 separate simulations including 412 isotopes and different starting energies. A considerable loss in performance was associated with the decrease of neutron population, and the best speedup was observed when thread divergence was most expressed. A realistic reactor assembly was also considered, runtimes of event-based GUARDYAN were slower in this case. Profiling the application revealed that poor performance was due to the inefficient implementation of the transition step, which should be addressed in order to better exploit vectorization gains.

KEYWORDS: vectorization, Monte Carlo, event-based

1. INTRODUCTION

GUARDYAN (GPU Assisted Reactor Dynamic Analysis) is a continuous energy Monte Carlo (MC) neutron transport code developed at Budapest University of Technology and Economics. It targets to solve time-dependent problems related to fission reactors with the main focus on simulating and analyzing short transients. The key idea of GUARDYAN is a massively parallel execution structure making use of advanced programming possibilities available on CUDA (Compute Unified Device Architecture) enabled GPUs (Graphics Processing Units). In order to maximize performance however, the architecture calls for some major deviations from a traditional MC code.

We have established two independent branches of the code, both designed to exploit the parallel capabilities of the GPU, although via quite different strategies. The first branch uses a very straightforward approach, that is the concurrent simulation of particle histories. One working unit is designated to simulate a particle history from birth to death, following the traditional history-based structure. This idea exploits the inherent parallelism of MC tracking, i.e. particle histories are independent of each other. On the other branch the code is vectorized, meaning that simultaneously executed operations are expected to be identical. While vectorization of the code would not mean much difficulty in case of deterministic methods, MC simulations are ill-suited for this task by the random nature of the process. Preserving a history-based structure is in this case unfeasible, thus an event-based strategy was implemented. A parallel MC calculation is called event-based when only particles undergoing the same event are simulated concurrently. In this case, one working unit is assigned to calculate the outcome of one event in a particle history. The tracking routine first assigns events (e.g. free-flight, fission, elastic scatter) to particles, creating stacks of particles undergoing the same event, then these stacks are processed separately.

2. PREVIOUS WORK

First successful attempts to vectorize MC calculations were made in the 1980s, following the appearance of efficient vector computers. Modification of the conventional Monte Carlo algorithm was discussed in detail by [1], elaborating on both general and specific aspects of vectorization. The first example of a vectorized general purpose MC code, RACER3D, reported speedups of at least 10 for a PWR (pressurized water reactor) setup [2]. Efficiency gain of a factor of 8-22 was found when implementing the vectorized MVP continuous energy MC code [3]. In a paper from 1987 [4], Martin summarizes several successful implementations and possible variants of the method (stack-driven version, zone-selection method), but raises concerns about the level of efficiency of vectorized MC algorithms on future computing platforms. And indeed, current studies indicate that the acceleration gain through vectorization may not be as emphasized as before, its merits are unclear. This is mainly due to the versatility of advanced architectures, and that many built-in (implicit) and explicit optimization tools support the efficient execution of the task.

In Monte Carlo particle transport methods, beside clusters of multicore nodes another viable option for high performance computing is the use of general purpose graphics processing units (GPGPUs) [5]. Vectorization of a MC code addresses the issue of what is called thread divergence in GPGPU terminology (different tasks are assigned to working units, resulting in the degradation of parallel performance). The event-based algorithm has already been implemented for GPU in several cases [6] [7]. However, recent studies point out that eliminating thread divergence may not be a decisive factor in the efficiency optimization of MC algorithms on the GPU. Although the occurrence of thread divergence was successfully reduced in [8], the vectorized version was found to be 10 times slower than the conventional algorithm. Memory latency issues and suboptimal arrangement of thread working load was accounted for the poor results. [9] suggests that a history-based algorithm can be significantly accelerated by only minor adjustments to the code. This is confirmed by [10], observing speedups of 2-4x over a 8-core CPU implementation. Following up on this idea, [11] reports that performance of a big-kernel history-based approach is comparable to the performance of the event-based version in a one dimensional setup with binary stochastic media. Thus the efficiency of vectorized MC simulations on GPUs is still up to debate. As the vectorization of a MC code requires nearly complete rewrite of the algorithm, production-level codes may suffer more than gain from vectorization (validation and verification work would be rendered), but new codes could efficiently exploit GPU capabilities [5]. The possibility of a hybrid simulation scheme is also worth to consider, when the event-based version is combined with the traditional history based method. Theoretical advancements on the prediction of parallel efficiency of event-based algorithms were recently developed in [12], a comparison of history- and event-based approaches in more complex geometries has not been discussed yet, this issue is addressed by this paper.

3. VECTORIZED MONTE CARLO ON CUDA ENABLED GPU

3.1. Parallel Execution Structure of GPUs

Currently GUARDYAN runs on a machine containing two Nvidia GeForce GTX 1080 cards, each with 8 GBytes of global memory and 5500 GFLOP/s single precision performance according to NBody GPU benchmark. The GTX 1080 cards are based on the Pascal architecture and have 2560 scalar working units (CUDA cores). These cores can launch warps of 32 concurrent threads, resulting in a theoretical maximum of 81920 parallel working units. The optimal number of concurrent threads may of course differ due to memory and arithmetic latency considerations. Thread management is implemented by organizing a desired number of threads into blocks, which are required to execute independently. This also ensures automatic scalability of the program, as blocks of threads can be scheduled on any multiprocessors of the device, yielding faster execution time when more multiprocessors are available. To better understand the execution structure of the GPU, thread hierarchy is presented on Fig. 1. Functions executed in parallel are called kernels in CUDA terminology. Kernels are launched by specifying the number of threads in a block, and the total number of blocks. In general, to choose the number of threads in a block as a multiple of warp size (32) is a good idea, however, CUDA offers an opportunity to maximize kernel performance automatically: calling the *cudaOccupancyMaxPotentialBlockSize* function for every kernel ensures optimal occupancy in terms of arithmetic intensity and memory latency.

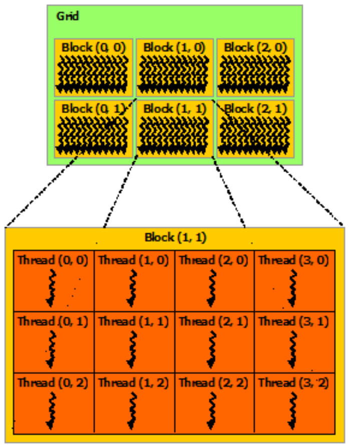

Figure 1: Thread hierarchy on the GPU

3.2. Memory Management

CUDA distinguishes six memory types: register, local, shared, texture, constant and global memory. Registers ensure the fastest memory access and are assigned to each thread. Global, constant and texture memory can be accessed by all threads, while the scope of shared memory is only a block. In exchange, it is much faster. Texture memory is not truly a distinct memory type, it only labels a part of global memory that is bound to texture. Textures are implemented with hardware interpolation, thus they would be ideal for storing cross section data. But due to random memory access patterns inherent in MC simulations, using cached memory is not advised in this case [7], thus cross sections are stored in global memory. A severe limitation for MC applications is the size of global memory, e.g. in GUARDYAN, nuclear data for one temperature occupies about 6GB on a card with global capacity of 8GB. Fig. 2 gives a summary of memory types available on GPUs. Fig. 2 also shows that memory transactions between GPU (device) and CPU (host) are carried out through reading and writing global memory. As the access of global memory is slow, these transactions can also take a considerable time, and can have a significant impact on overall performance. Notice, that if the simulation structure is changed (e.g. a history-based algorithm is vectorized), register use, global memory reads and host-device communication will show different behavior, also influencing runtime. Thus the performance gain from vectorization will be obscured.

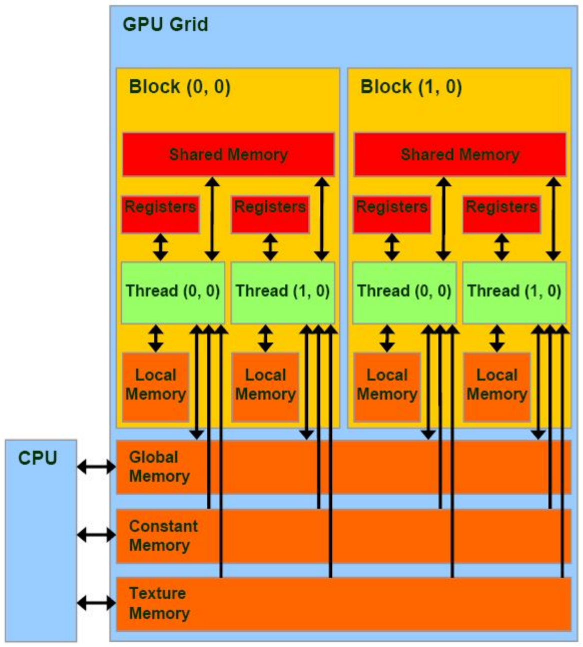

Figure 2: CUDA memory types

3.3. Thread Divergence

When implementing a MC neutron transport code special attention must be paid to choosing the right kernels, as the slowest working thread will determine the efficiency of parallelization. All other threads in a warp must wait for the thread finishing last. Loops, conditional and branching statements lead to thread divergence, an uneven distribution of work-load. This issue is targeted by the vectorization of the code, i.e. the event-based Monte Carlo simulation. In event-based GUARDYAN this is implemented by distinguishing kernel functions for different types of events instead of just one ”big” kernel (as in the history-based GUARDYAN). When calling these kernels, threads of a warp executing the same operations on particles (the same event is simulated) do not branch, whereas in a history-based simulation threads may easily diverge as one particle may be in a transition step while the other scatters or induces fission. Branching statements in a warp are executed serially in CUDA: an if-else statement is executed for all warps in two cycles (both branch is executed one after the other). When the ”if” branch is executed, the threads that do not satisfy the condition (would diverge to the ”else” branch) are flagged and perform a NOP (no operation). This results in the degradation of parallel performance. However, it does not necessarily mean that the vectorized version of the code will execute faster. One reason was given in the previous section, regarding memory management issues. Neither should we neglect that the compiler also does some optimization to reduce the penalty due to thread divergence [13], among these are well-known tools like warp-voting and predication, but CUDA uses optimization tools that are hidden from the programmer [14].

3.4. Event Based Version

We implemented a basic event-based version of the code by assigning new CUDA kernels to each energy law. A transition kernel is executed on all particles performing a Woodcock step and the selection of the next collision event. The issue of thread divergence is then addressed by sorting neutrons according to the type of the upcoming interaction, and calling the new kernel functions on the appropriate particles. Thus the same energy law is applied to all neutrons simulated by a kernel. Sorting is done by calling the *sort_by_key* function of the *thrust* library.

4. PERFORMANCE OF EVENT BASED TRACKING

4.1. Test Case #1 - Validation Setup

GUARDYAN has been recently validated against MCNP6 in a simplified setup assuming a mono-energetic neutron source inside a homogeneous sphere. $4 \cdot 10^6$ neutrons were launched at energies $0.01eV$, $1eV$, $1keV$, $1MeV$ and $18MeV$, and tracked until either leaking out of the sphere or exceeding time boundary. The simulation was carried out for 412 isotopes and was used for validation of the code comparing the spectrum of leaking neutrons to MCNP6 results. Cross section library ENDF/B-VII.1 was used assuming temperature of $293.6K$. Regarding our investigation of event-based tracking, wall-time was measured for both history-based (T_H) and event-based (T_E) simulations. On Fig. 3, histograms of simulation speedup are plotted for all starting energies. Speedup is simply defined by

$$\text{Speedup} = \frac{T_E}{T_H}$$

i.e. the ratio of wall-times. Fig. 3 shows that vectorization of the code resulted in faster execution time in most cases. Typical speedup was around 1.5-2, but longer simulation time was observed mainly when starting energy is below $1MeV$. The efficiency loss was experienced in case of isotopes with high probability for fission around the starting energy. When the starting energy is low, neutrons released in fission take on much higher velocity than starters, thus leaking out of the system very fast. As a result, significant part of computational effort was spent on a few neutrons bouncing around in the system. Population drop caused vectorization gain to be cancelled due to the computational overhead of event-based tracking (particles need to be sorted by event type). On higher starting energies, no considerable speedup was observed in case of elements with low atomic numbers, the improvement from vectorization was more expressed when heavy elements were present. This is due to that the outgoing energy and angle of a neutron scattered on a light isotope are derived by simple laws of collision mechanics, while more complicated energy laws are applied when heavier isotopes are present [15]. In GUARDYAN beside elastic scatter only ACE law 3 (inelastic discrete-level scattering) was used in the former case, and ACE law 4 (and 44) was additionally used in the latter. ACE law 4 represents a continuous tabular distribution, the outgoing energy is given as a probability distribution for every incoming energy [16]. This sampling procedure takes considerably more time, contributing to thread divergence, and resulting in substantial efficiency boost for event-based tracking.

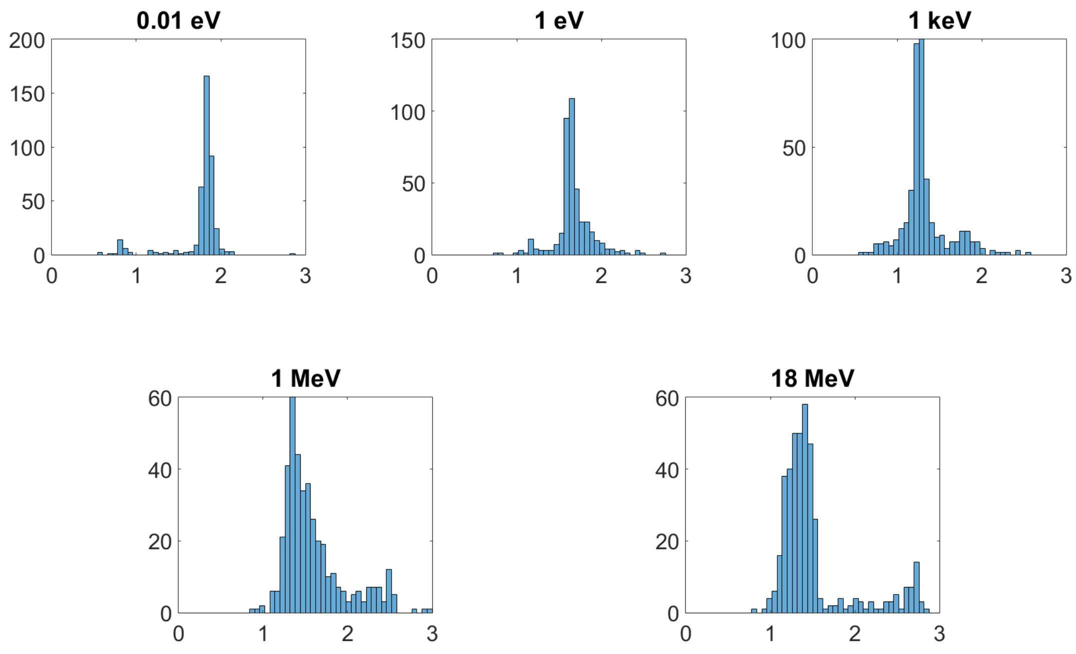

Figure 3: Frequency of simulation speedup due to vectorization in case of different starting energies

4.2. Test Case #2

Our investigations in the validation setup pointed out that the efficiency of event-based tracking is significantly reduced when the neutron population decreases. Highest speedups were detected when the simulation used several sampling laws associated with various computational cost for calculating the outgoing energy and angle of a neutron. In test case #2, we assumed an inhomogeneous medium, depicted in Fig. 4. The geometry contained 61 uranium dioxide rods embedded in a light water sphere. Validation of both history-based and event-based version of GUARDYAN was carried out for this problem, measuring the flux of escaping neutrons from the sphere during 9 time intervals (Fig. 5). We experienced no deviation from the reference solution outside the statistical variance.

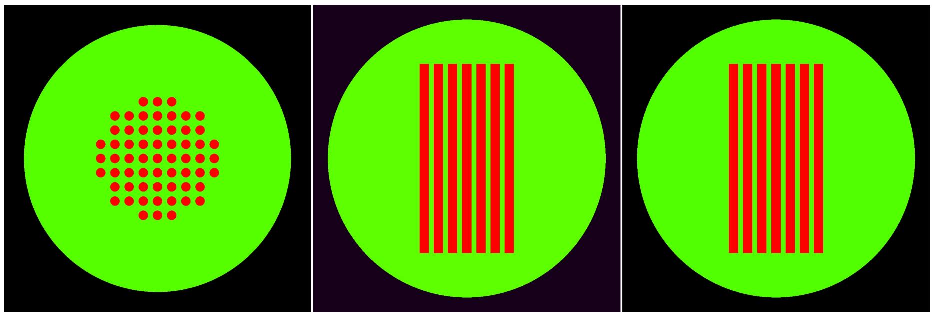

Figure 4: Geometry of the inhomogeneous sample problem

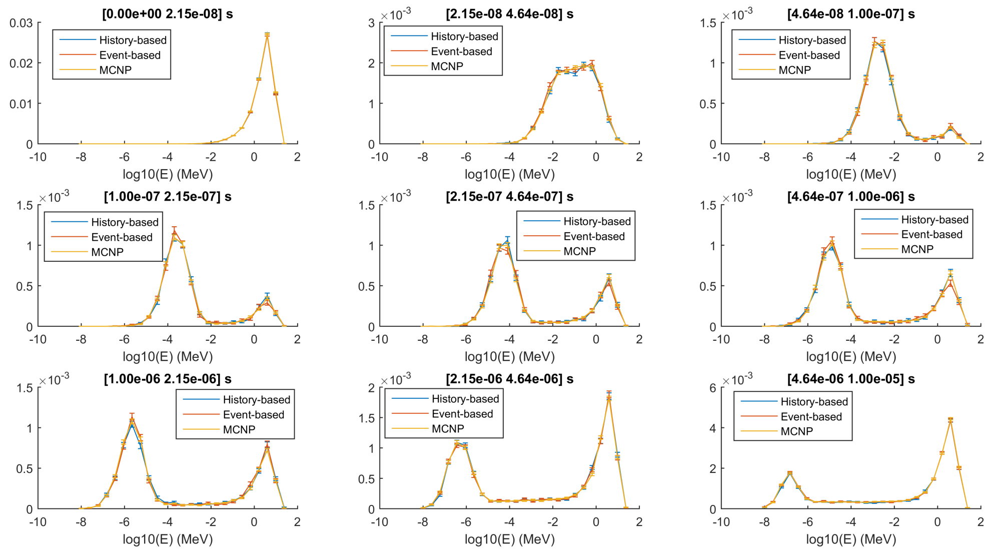

Figure 5: Validation of history- and event based GUARDYAN against MCNP6. The graphs show the spectrum of particles escaping the sphere in a given time interval

Table 1 shows execution times measured during the simulation of neutron transport in the inhomogeneous sample problem. Wall-times of history-based and event-based versions show no significant difference, the vectorized code performed slightly better. It must be noted that although speedups are also shown relative to MCNP runtime, these should not be considered conclusive, as the MCNP simulation ran on a single core, also, GUARDYAN is highly underoptimized. Nevertheless a factor of 6 was observed in performance over the single core MCNP simulation, which is poor compared to the efficiency gain due to CUDA implementation in Ref. [11] (however, the test geometry was also different). This supports that GUARDYAN should be further improved, we believe that introducing new variance reduction techniques and optimizing algorithms should definitely help to boost the simulations.

Table 1: Parallel performance for the inhomogeneous sample problem

	MCNP6	GUARDYAN history-based	GUARDYAN event-based
Wall-time (min)	39.9	6.73	6.22
Speedup	—	5.93	6.41

4.3. Light Water Reactor Assembly

GUARDYAN is capable of simulating neutron transport in complex geometries, but is has yet to be further developed in order to provide quantitative results. The event-based version was tested on the geometry of the training reactor at Budapest University of Technology and Economics, shown in Fig. 6.

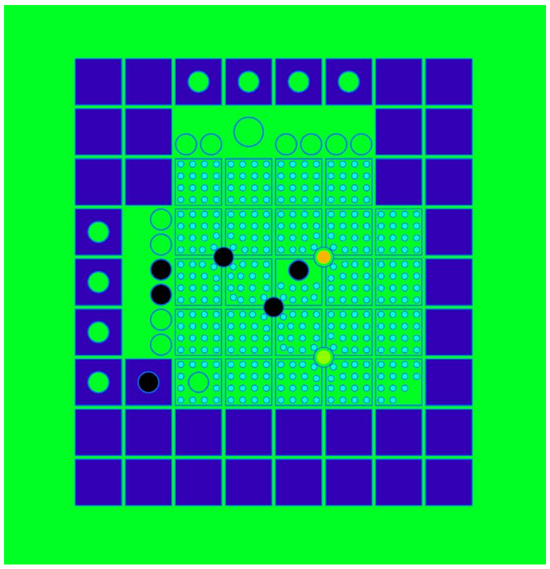

Figure 6: Zone of the training reactor at Budapest University of Technology and Economics

We experienced, that the vectorized code ran 1.5x slower than the history-based algorithm. To better understand the underlying reasons we looked into the kernel execution times. In case of the event based version of GUARDYAN, every energy law was implemented in a separate kernel, thus an application profiling tool is able to reveal which task consumed most resources. Inspecting the profile shown in Fig. 7, several conclusions can be made:

- The main part of the execution time is due to calling the ”transition kernel”. This function transports a particle to the next collision site, and performs the selection of reaction type for that particle. Long calculation time is most likely caused by the Woodcock method used for path length selection (a phenomenon termed the heavy absorber problem) and slow energy grid search algorithms implemented in GUARDYAN. A possible solution for the heavy absorber problem may be solved according to our recent investigation of biased Woodcock algorithms [17].

- Memory transaction costs are much greater than computational costs of simulating different reactions. The "CUDA memcpy DtoH" and "CUDA memcpy HtoD" tasks stand for the communication between host and device, taking up more simulation time than simulating elastic scatter and ACE laws.
- The "Thrust sort" kernel includes all computational overhead that is associated with event-based tracking. Note, that sorting is done two orders of magnitudes faster than memory transactions.

Fig. 7 indicates that history based tracking may be more effective because most of the calculation time is due to calling one kernel (called "transition kernel") which is applied to all particles before every collision. In order to execute the simulation of any type of reaction, the event based version must wait for the transition step to end for all particles. On the other hand, the history-based simulation can go on unsynchronized, i.e. threads may diverge (one may execute a transition step while the other simulates a collision), but threads do not need to wait for others to proceed. By optimization of the transition step,

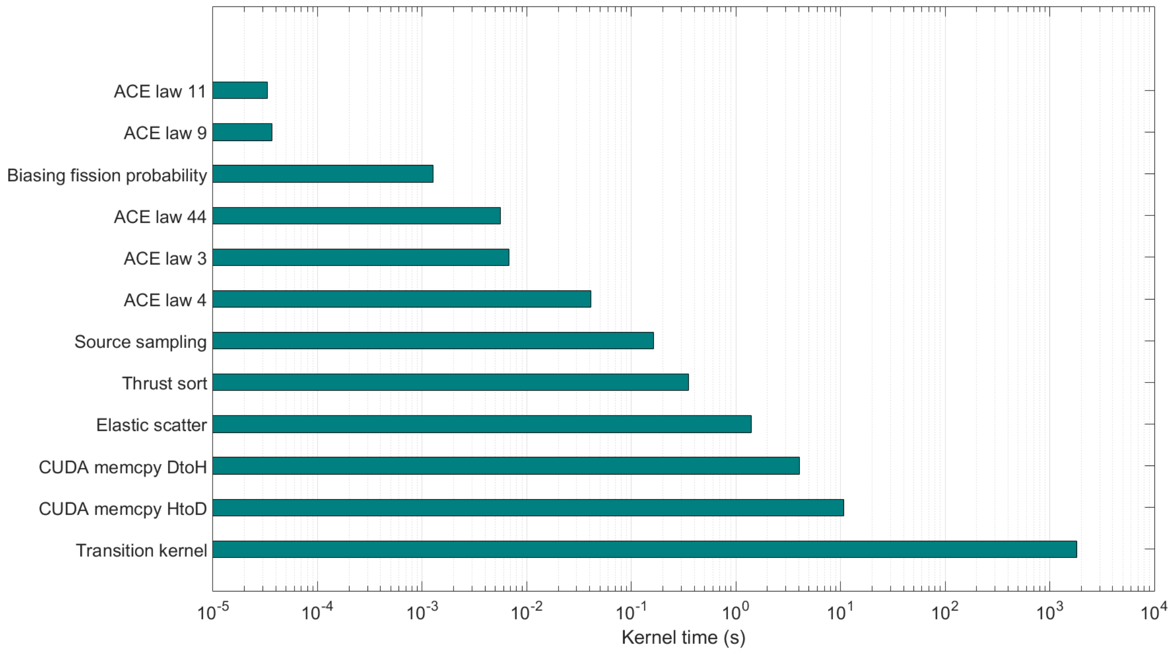

Figure 7: Profile of the application transporting neutrons in the LWR assembly

5. CONCLUSIONS

The paper investigates the performance gain due to the vectorization of a Monte Carlo code on modern computer architectures. A GPU accelerated dynamic MC code, GUARDYAN, was successfully vectorized and analyzed in different scenarios. In the recent validation of GUARDYAN a homogeneous setup was studied, in which the efficiency of event-based tracking was also measured. Speedups of 1.5-2 were typically observed in 2060 separate simulations including 412 isotopes and different starting energies. A considerable loss in performance was associated with the decrease of neutron population, and the best speedup was observed when thread divergence was most expressed, i.e. different type of sampling laws were applied with various computational cost. Regarding an inhomogeneous sample problem, no substantial difference was discovered between the history-based and event-based versions. A realistic assembly was also considered. Runtimes of event-based GUARDYAN were slower in this case, but profiling the application revealed that poor performance was due to the inefficient implementation of the transition step, indicating that further developments should be made to improve GUARDYAN. Future work should focus on reducing the computational cost of transporting particles between collisions (including the selection of reaction type), if major improvements could be achieved, then the vectorized version can also be considered as an alternative tracking method.

ACKNOWLEDGEMENTS

This work has been carried out in the frame of VKSZ_14-1-2015-0021 Hungarian project 260 supported by the National Research, Development and Innovation Fund.

REFERENCES

[1] F. B. Brown and W. R. Martin. "Monte Carlo methods for radiation transport analysis on vector computers." *Progress in Nuclear Energy*, **volume 14**(3), pp. 269–299 (1984).

[2] F. Brown. "Vectorization of three-dimensional general-geometry Monte Carlo." *Trans Am Nucl Soc*, **volume 53**(CONF-861102-) (1986).

[3] T. Mori, M. Nakagawa, and M. Sasaki. "Vectorization of Continuous Energy Monte Carlo Method for Neutron Transport Calculation." *Journal of Nuclear Science and Technology*, **volume 29**(4), pp. 325–336 (1992).

[4] W. Martin and F. B. Brown. "Status of Vectorized Monte Carlo for Particle Transport Analysis." **volume 1** (1987).

[5] F. B. Brown. "Recent advances and future prospects for Monte Carlo." Technical report, Los Alamos National Laboratory (LANL) (2010).

[6] R. Bleile, P. Brantley, S. Dawson, M. O'Brien, and H. Childs. "Investigation of Portable Event-Based Monte Carlo Transport Using the NVIDIA Thrust Library." Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2016).

[7] 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*, **volume 77**, pp. 176–193 (2015).

[8] X. Du, T. Liu, W. Ji, X. G. Xu, and F. B. Brown. "Evaluation of vectorized Monte Carlo algorithms on GPUs for a neutron eigenvalue problem." In *Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013), Sun Valley, Idaho, USA*, pp. 2513–2522 (2013).

[9] A. Scuderio. "Monte Carlo Neutron Transport: Simulating Nuclear Reactions One Neutron at a Time." In *GPU Technology Conference, San Jose, California, May, 2014* (2014).

[10] S. P. Hamilton, T. M. Evans, and S. R. Slattery. "GPU Acceleration of History-Based Multigroup Monte Carlo." *Trans Am Nucl Soc*, **volume 115**, p. 527530 (2016).

[11] P. S. Brantley, R. C. Bleile, S. A. Dawson, N. A. Gentile, M. S. McKinley, M. J. OBrien, M. M. Pozulp, D. F. Richards, D. E. Stevens, J. A. Walsh, and H. Childs. "LLNL Monte Carlo Transport Research Efforts for Advanced Computing Architectures." In *Proceedings of International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering (M&C 2017), Jeju, Korea* (2017).

[12] P. K. Romano and A. R. Siegel. "Limits on the efficiency of event-based algorithms for Monte Carlo neutron transport." *Nuclear Engineering and Technology*, **volume 49**(6), pp. 1165 – 1171 (2017). Special Issue on International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering 2017 (M&C 2017).

[13] P. Bialas and A. Strzelecki. "Benchmarking the cost of thread divergence in CUDA." In *International Conference on Parallel Processing and Applied Mathematics*, pp. 570–579. Springer (2015).

[14] B. Coon, J. Nickolls, L. Nyland, P. Mills, and J. Lindholm. "Indirect function call instructions in a synchronous parallel thread processor." (2012). US Patent 8,312,254.

[15] J. Hoogenboom. "Consequences of inelastic discrete-level neutron-collision mechanics for inelastic continuum scattering." *Annals of Nuclear Energy*, **volume 10**(1), pp. 19–29 (1983).

[16] J. F. Briesmeister et al. "MCNP-A general Monte Carlo N-particle transport code." *Version 4C, LA-13709-M, Los Alamos National Laboratory*, p. 2 (2000).

[17] D. Legrady, B. Molnar, M. Klausz, and T. Major. "Woodcock tracking with arbitrary sampling cross section using negative weights." *Annals of Nuclear Energy*, **volume 102**, pp. 116–123 (2017).