Performance and Accuracy of WARP - A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs

Ryan M. Bergmann*, Kelly Rowland, Nikola Radnović, Jasmina L. Vujić

Department of Nuclear Engineering, 4155 Etcheverry Hall, University of California - Berkeley, Berkeley, CA 94703-1730

Abstract

In recent supercomputers, general purpose graphics processing units (GPGPUs) are a significant faction of the supercomputer's total computational power. GPGPUs have different architectures compared to central processing units (CPUs), and for Monte Carlo neutron transport codes used in nuclear engineering to take advantage of these coprocessor cards, transport algorithms must be changed to execute efficiently on them. WARP is a continuous energy Monte Carlo neutron transport code that has been written to do this. The main thrust of WARP is to adapt previous event-based transport algorithms to the new GPU hardware; the algorithmic choices for all parts of which are presented in this paper. It is found that remapping history data references increases the GPU processing rate when histories start to complete. The main reason for this is that completed data are eliminated from the address space, threads are kept busy, and memory bandwidth is not wasted on checking completed data. Remapping also allows the interaction kernels to be launched concurrently, improving efficiency. The OptiX ray tracing framework and CUDPP library are used for geometry representation and parallel dataset-side operations, ensuring high performance and reliability.

Keywords: Monte Carlo, Neutron Transport, GPU, CUDA, CUDPP,

*Corresponding author. Tel.: +1 320 333 3089.

Email addresses: ryanmbergmann@gmail.com (Ryan M. Bergmann), krowland@berkeley.edu (Kelly Rowland), radnovicn@gmail.com (Nikola Radnović), vujic@nuc.berkeley.edu (Jasmina L. Vujić)

1. Introduction

Graphics processing units, or GPUs, have gradually increased in computational power from the small, job-specific boards of the early 1990s to the programmable powerhouses of today. Compared to more common central processing units, or CPUs, GPUs have a higher aggregate memory bandwidth, much higher rate of floating-point operations per second (FLOPS), and lower energy consumption per FLOP. Because one of the main obstacles in exascale computing is power consumption, many new supercomputing platforms are gaining much of their computational capacity by incorporating GPUs into their compute nodes. Since CPU-optimized parallel algorithms are not directly portable to GPU architectures (or at least not without losing substantial performance), particle transport codes need to be rewritten to execute efficiently on GPUs. Unless this is done, reactor simulations cannot take full advantage of these new supercomputers. In this work, the algorithmic choices used in WARP and their consequences are presented. The ultimate accuracy of WARP, comparisons against other neutron transport codes, and benchmarking are not presented here. This work only concerns algorithmic decisions based on self-comparison. WARP benchmarking, accuracy and speedup comparison with other Monte Carlo codes will be presented in a companion paper.

WARP, which can stand for "Weaving All the Random Particles," is a three-dimensional (3D) continuous energy Monte Carlo neutron transport code developed at UC Berkeley to efficiently execute on a CPU/GPU platform. WARP accelerates Monte Carlo simulations while preserving the benefits of using the Monte Carlo method, namely, that very few physical and geometrical simplifications are applied. WARP is able to calculate multiplication factors, flux tallies, and fission source distributions for time-independent problems, and can run in both criticality or fixed source modes. WARP can currently transport neutrons in unrestricted arrangements of parallelepipeds, hexagonal prisms, cylinders, and spheres.

The impetus for developing WARP was the research done by Martin and Brown in 1984 [**?**] for Monte Carlo and by Vujić and Martin in 1991 [**?**] for collosion probability method. In the 1984 paper, a method for mapping the Monte Carlo problem onto SIMD (single instruction multiple data) vector

computers is described. SIMD is an execution model some processors use in order to lower the number of instructions needed per amount of computation done, which increases both power and computational efficiency [**?**]. SIMD requires the same instructions to be carried out over every element in a concurrently-processed data vector. The essential idea in the 1984 paper is to bank the neutrons into vectors based on their required operation. If a neutron is scattering, its data is placed in the scattering buffer. If a neutron needs to do a surface crossing, it is put in the crossing buffer, and so on. Once a buffer becomes full, it is processed in a SIMD fashion by the vector computer. Processing the neutrons makes the buffers contain non-uniform reactions, however, and a "shuffle" operation is done that actually moves data back into contiguous blocks based on the reaction type.

This new approach was named "event-based" Monte Carlo, since the neutron events are tracked and processed as a group [**?**]. This was a very different way of performing a Monte Carlo simulation at the time. Almost all computers were strictly serial, and SIMD lanes were only available in supercomputers. Therefore, the pervasive method was the "task-based" method in which neutrons are tracked for their entire lifetime in series. Since GPUs are massively parallel and rely on SIMD, an event-based algorithm seems to be the appropriate approach to GPU-accelerated neutron transport.

WARP uses an event-based algorithm, but with some important differences. Vectorizing the Monte Carlo algorithm should allow for efficient GPU execution, but a paper by Zhang [**?**] also shows that by remapping data references, the thread divergence in GPU warps can be minimized. Moving data is expensive, so WARP uses a remapping vector of pointer/index pairs to direct GPU threads to the data they need to access. The remapping vector is sorted by reaction type after every transport iteration using a high-efficiency parallel radix sort, which serves to keep the reaction types as contiguous as possible and removes completed histories from the transport cycle. Sorting reduces the amount of divergence in GPU "thread blocks," keeps the SIMD units as full as possible, and eliminates using memory bandwidth to check if a neutron in the batch has been terminated or not. Using a remapping vector means the data access pattern is irregular, but this is mitigated by using large batch sizes where the GPU can effectively eliminate the high cost of irregular global memory access.

WARP sets itself apart from any previous endeavors in its breath of scope and its novel adaption of the event-based transport algorithm. Previous codes have also either used synthetic, simplified, and/or incomplete nuclear data [**?**

? ?]. WARP loads standard data files and accurately simulate each reaction type specified in the data. WARP also uses a flexible, scalable, and optimized geometry representation where previous studies have used simplified and restricted geometry models [?]. Previous works have examined event-parallel algorithms, but have not parallelized them effectively and therefore did not see the benefits of adopting such an algorithm on a GPU [?]. WARP uses highly-parallelized algorithms and slightly modify the original vision of the event-based algorithm to better suit execution on the GPU. The previous event-based algorithms tried to implement a "shuffle" operation where neutron data was actually sorted into reaction-contiguous blocks [?], or used small, synthetic nuclear data and were not able to capture the effects of loading large nuclear datasets [?]. Other studies have been done that offload specific parts of the neutron transport routine to the GPU, such as that done on RMC at Tsinghua University [?]. They were able to obtain a 113x speedup over a CPU version without a flux tally and 36x speedup with a flux tally [?]. Again, one group cross sections were used.

WARP also changes the unionized energy grid data format to reduce the number of data loads needed to scan cross sections. Instead of storing a matrix of pointers indexed by reaction type and energy, WARP stores three matrices. The first contains cross section values, the second contains pointers to angular distributions, and a third contains pointers to energy distributions. This linked list type of layout increases memory usage, but lowers the number of data loads that are needed to determine a reaction by eliminating a pointer load to find a cross section value.

Optimized, high-performance GPU code libraries are also used by WARP wherever possible. The CUDA performance primitives (CUDPP) library is used to perform the parallel reductions, sorts and sums, the CURAND library is used to seed the linear congruential random number generators, and the OptiX ray tracing framework is used for geometry representation [? ? ?]. OptiX is a highly-optimized library developed by NVIDIA that automatically builds hierarchical acceleration structures around user-input geometry so only surfaces along a ray line need to be queried in ray tracing [?]. WARP also performs material and cell number queries with OptiX by using a point-in-polygon like algorithm. The reaction sampling routines have been coded in CUDA according to the appropriate ENDF laws.

WARP is designed to read ACE-formatted data [?] and perform all reaction types as prescribed by the data. ACE is an acronym for "A Compact ENDF," and is the format which MCNP and Serpent both read [? ?]. ACE

data is loaded with the PyNE package [**?**] and passed to the main C++ code via the Python C application programming interface (API). WARP uses a Serpent-like unionized energy grid to regularize data access (Serpent is a Finnish Monte Carlo reactor physics code). A unionized energy grid is one where the individual energy grids of every isotope used in a simulation have been combined into a single energy vector. The cross section vectors are then interpolated on this larger unionized energy grid [**?**]. The host-side code in WARP is written in C/C++. Single precision floating point numbers are used throughout in order to realize the full computational capacity of the GPU and to allow simulations to be carried out on more affordable and higher clocked GeForce cards [**?** **?**]. Using single precision numbers may lead to inaccuracies when there are very dilute isotopes or very rare reactions reactions present, as roundoff error may make their contributions zero. Buffer overflow and roundoff error in the tallies may also be a problem with single precision, but this can be mitigated by accumulating the tallies frequently in a double precision vector. Roundoff error may also be problematic in calculating the multiplication factor, but this can also be mitigated by accumulating the integer secondary neutron yield values into a large host-side variable. If double precision data is found to be needed, WARP can easily be changed, and doing so may be an interesting experiment in the future.

Section 2 discusses how OptiX is used to perform surface detection and query the current material a neutron is traveling through. The testing done to determine the optimal OptiX configuration is also discussed. Section 3 deals with the way the nuclear data is loaded, reformatted to use a unionized energy grid, and the details about data layout and access patterns. Section 4 details each individual CUDA kernel (i.e. GPU program) used by WARP to process the neutron histories. A Section 5 presents the results of the OptiX testing and performance improvements attributed to reference remapping, and conclusions based on the results are mode in Section 6.

2. Ray Tracing and Geometry Representation with OptiX

2.1. Instancing

2.2. Test Geometries

3. Unionized Cross Section Data Layout

4. CUDA Kernels and Data-Parallel Tasks

4.1. Criticality Source

5. Results

5.1. OptiX Scaling Study Results

5.2. The Effects of Reference Remapping

6. Conclusions

WARP has shown that GPUs are an effective platform for performing Monte Carlo neutron transport with continuous energy cross sections. Currently, WARP is the most detailed and feature-rich program in existence for performing continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs, but compared to production codes like Serpent and MCNP, WARP has limited capabilities. Despite WARP's lack of features, its novel algorithm implementations show that high performance can be achieved on a GPU despite the inherently divergent program flow and sparse data access patterns. Remapping threads to active data is an effective way of raising the processing rate when the number of active neutrons becomes small; this also reduces thread divergence in reaction kernels. Using a radix sort to do the remapping is effective since it segregates reactions into contiguous blocks, efficient since it can be done in place and in $O(kN)$ time, and can eliminate completed data from being accessed if slight modifications to the standard reaction number encodings are made.

Most of the performance gain in remapping data references comes from being able to launch grids that are sized for only the active data rather than the entire dataset for both global and reaction kernels. A non-remapping algorithm does not keep track of where active data is, and therefore must launch a grid that covers the entire dataset. When the number of active neutrons drops below about 30% of the initial number, the overhead and memory bandwidth cost of launching these extra threads, which only load a "done" bit and return, is more than the cost of performing the radix sort and edge detection. The majority of the transport iterations occur while there

are less than 30% of the initial neutrons left, and remapping references is usually worthwhile.

Using the NVIDIA OptiX ray tracing framework was also shown to be an effective way to handle the geometry representation in WARP. OptiX is flexible, allows attachment of material and cell number to individual geometric primitives, can perform surface detection with a randomly-distributed and directed dataset, can incorporate the remapping vector created by a radix sort, and is fast enough to be used in WARP. The acceleration structures that OptiX can automatically build over the scene geometry was the initial reason for using it, and it was determined that the BVH builder and traverser provide the best performance as does using mesh primitive instancing rather than a transform node approach. The number of objects present in the scenes in reactors is small compared to many rendering scenes, and the SBVH acceleration structure does not perform as well. This is presumably due to some additional overhead related to traversing the objects that is not offset when few objects (less than a few hundred thousand) are present in the scene. Primitive instance provides better performance since using transform nodes requires traversing a deeper geometry tree, which also has more (redundant) data associated to it.

As mentioned in Section 1, a forthcoming companion paper is planned that will benchmark WARP against Serpent and MCNP using identical cross section libraries and problem geometries. In the paper, the relative accuracy and speed of WARP will be determined by comparing it to these production-level Monte Carlo neutron transport codes.

Acknowledgements

This research is based upon work partially supported by the U.S. Department of Energy National Nuclear Security Administration under Award Number DENA0000979 through the Nuclear Science and Security Consortium: http://nssc.berkeley.edu.

Disclaimer