Delta-tracking in the GPU-accelerated WARP Monte Carlo Neutron Transport Code

by

Kelly L. Rowland

A report submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Engineering - Nuclear Engineering

in the

Graduate Division

of the

University of California, Berkeley

Spring 2015

The report of Kelly L. Rowland, titled Delta-tracking in the GPU-accelerated WARP Monte Carlo Neutron Transport Code, is approved:

Chair	________________________________	Date	______________
	________________________________	Date	______________
	________________________________	Date	______________

University of California, Berkeley

Delta-tracking in the GPU-accelerated WARP Monte Carlo Neutron Transport Code

Contents

List of Figures

List of Tables

Acknowledgments

This material is based upon work supported under an Integrated University Program Graduate Fellowship.

This work was partially supported by the Department of Energy National Nuclear Security Administration under Award Number(s) DE-NA0000979: The National Science and Security Consortium (NSSC), http://nssc.berkeley.edu/.

Chapter 1

Introduction

1.1 Motivation

Graphics processing units (GPUs) have gradually increased in computational power from small, job-specific boards to programmable powerhouses. Compared to more common central processing units (CPUs), GPUs have a higher aggregate memory bandwidth, much higher floating-point operations per second (FLOPs), and lower energy consumption per FLOP [1].

As we approach exascale computing, many institutions are investing in GPU hardware to help achieve these peak speeds. In the United States, Oak Ridge National Laboratory (ORNL) and Lawrence Livermore National Laboratory (LLNL) are both acquiring high-performance computing systems that include NVIDIA GPU architectures [2, 3]. The computer system to be built at the Oak Ridge Leadership Computing Facility will be named "Summit" and will consist of nodes with multiple IBM POWER9 CPUs and NVIDIA Volta accelerators. The architecture will use a coherent memory space that includes high bandwidth memory on the GPUs and a high speed NVLink interconnect between the POWER9 CPU and Volta GPUs [2]. "Sierra", the next advanced technology high-performance computing system to be introduced into LLNL's High Performance Computing Complex, will also incorporate IBM POWER and NVIDIA Volta processors [3]. Additionally, it is estimated that Sierra will be about five times more power efficient than its predecessor, Sequoia, while providing four to six times the sustained performance and five to seven times the workload performance.

Supercomputing systems that use GPUs to accelerate computation are competitive in terms of speed when compared to other types of supercomputers. The TOP500 project uses a benchmark of ability to solve linear systems of equations in dense matrix form to rank supercomputers. The most recent TOP500 list was published in November 2014 [4] and demonstrated that supercomputers using GPU acceleration can achieve peak speeds. The No. 2 and No. 6 systems both use NVIDIA GPUs to accelerate computation. Overall, fifty of the top 500 systems use NVIDIA chips for acceleration [4].

However, transitioning to using these GPU-accelerated supercomputers will require substantial time and effort. For many algorithms, CPU-optimized parallel algorithms are not directly portable to GPU architectures. In the field of nuclear engineering research and simulation specifically, neutral particle transport codes must be rewritten to execute efficiently on GPUs [1]. In particular, the field of nuclear engineering uses particle transport codes for reactor analysis, criticality safety, nuclear security and nonproliferation, detection, and shielding calculations.

Currently, there are not any general purpose Monte Carlo codes that execute on GPUs. Two prominent codes used for neutron transport calculations in nuclear reactor analysis are the Monte Carlo N-Particle (MCNP) code and the Serpent code. Both pieces of software use Monte Carlo methods to track neutron histories and perform criticality calculations on CPU-based computer architectures [5, 6]. At this time, however, there are no plans to adapt the MCNP code for use on GPUs [7], providing incentive for nuclear engineering researchers to begin developing their own Monte Carlo neutron transport software for efficient use on GPUs.

1.2 WARP

WARP, which can stand for "Weaving All the Random Particles", is a three-dimensional (3D), continuous energy, Monte Carlo neutron transport code originally developed by Dr. Ryan M. Bergmann at UC Berkeley to efficiently execute on a CPU/GPU platform [1]. 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 currently transports neutrons in unrestricted arrangements of spheres, cylinders, parallelpipeds, and hexagonal prisms [1] and is able to entertain both vacuum and reflecting (specular) boundary conditions.

What sets WARP apart from previous, somewhat similar endeavors is its breadth of scope and novel adaptation of the event-based Monte Carlo algorithm. Previous codes have been limited to restricted nuclear data or simplified geometry models, where WARP instead loads standard data files and uses a flexible, scalable, optimized geometry representation [1]. WARP uses a suite of highly-parallelized algorithms and employs a modified version of the original event-based algorithm that is better suited to GPU execution. For full detail, please see the paper by Bergmann and Vujić (2015) [1].

Chapter 2

Background

2.1 Delta-tracking theory

The delta-tracking method was introduced by Woodcock in the 1960s. Although widely known, the method has not been commonly used as the primary transport method in general purpose Monte Carlo codes, probably because of its limitations discussed below [8].

In Monte Carlo neutron transport, we track anything that can happen to neutrons in a system: collisions and their outcomes, surface crossings, etc. To do that, we need to track how these particles move in that system. This starts with determining the number of path lengths they cover between events. The number of path lengths traveled by a given neutron is sampled from an exponential distribution using

$$\ell = \frac{-\log \xi}{\Sigma_{\mathrm{tot,m}}(E)}, \tag{2.1}$$

where ξ is a uniformly-distributed random variable on the unit interval and $\Sigma_{\mathrm{tot,m}}$ is the macroscopic total cross section of the material m in which the neutron is located. The total cross section characterizes the neutron collision probability per unit path length traveled in the medium.

However, the sampled path length is not statistically valid if the neutron crosses a material boundary; the flight is stopped at the boundary surface and a new path length is sampled with the new material total cross section. This is the main principle of conventional ray tracing [8] and is what is done in WARP with the OptiX framework, as discussed below.

The idea behind delta-tracking is to effectively homogenize the total cross sections such that the sampled path lengths are valid over the entire geometry. Consider the concept of the “virtual collision”, a scattering reaction that preserves neutron energy and direction. Since virtual collisions have no impact on the final outcome, any number of them can “happen” in the random walk. Therefore, the total cross section of a material can be adjusted by adding an arbitrary virtual collision cross section, $\Sigma_{0,\mathrm{m}}(E)$ [8]:

$$\Sigma'_{\mathrm{tot,m}}(E) = \Sigma_{\mathrm{tot,m}}(E) + \Sigma_{0,\mathrm{m}}(E). \tag{2.2}$$

Because all material cross sections can now be freely and arbitrarily increased, a *majorant* cross section can be assigned to represent the effective total (physical + virtual) collision probability in all materials along a given neutron's trajectory:

$$\begin{aligned}\Sigma_{\text{maj}}(E) &= \Sigma'_{\text{tot,1}}(E) = \Sigma'_{\text{tot,2}}(E) = \ldots = \Sigma'_{\text{tot,M}}(E) \\ &= \max\{\Sigma_{\text{tot,1}}(E), \Sigma_{\text{tot,2}}(E), \ldots, \Sigma_{\text{tot,M}}(E)\}\ ,\end{aligned} \tag{2.3}$$

where M is the number of materials along a neutron's flight path.

Path lengths sampled using the global majorant are statistically valid in all materials, effectively homogenizing the material total cross sections and eliminating the need to calculate surface intersection distances [8]. Instead, an additional step is included in the tracking routine for handling virtual collisions. Rejection sampling is carried out for each collision, and the collision point is accepted with probability

$$P_{\text{col}}(E) = \frac{\Sigma_{\text{tot,col}}(E)}{\Sigma_{\text{maj}}(E)}. \tag{2.4}$$

If the point is rejected, the collision is considered virtual and the random walk continues by sampling a new path length.

The inherent advantage of delta-tracking is that the neutron random walk can be continued over material boundaries without stopping the walk at each surface crossing. The geometry routine reduces to determining the material in which the collision point resides, which is computationally less expensive than calculating the surface distances when running a simulation on traditional CPU hardware [8]. The simplified geometry routine is advantageous in that delta-tracking is often faster than ray tracing in complex geometries, and complicated objects and surface types are easier to handle with a delta-tracking algorithm [8].

One shortcoming of the delta-tracking method arises when material total cross sections differ greatly. A representative example of this is a light water reactor (LWR) fuel assembly that contains localized heavy absorbers (such as control rods or burnable poison rods) [8]. Although the absorber itself occupies a relatively small space physically, its large cross section dominates the majorant at low neutron energies. This causes the neutron random walk to be cut into many short tracks, wasting computing time by continually incurring virtual collisions and resampling.

Additionally, using the delta-tracking method necessitates the use of the collision flux estimator rather than the track-length flux estimator, which is generally considered a drawback for implementation in traditional Monte Carlo codes [8]. The track-length estimation of the reaction rate integral can be written in simplest analog form as

$$R_{\text{tle}} = \sum_{i=1}^{I} \ell_i f_i, \tag{2.5}$$

where ℓ is the neutron track length, f is the response function, and the summation is over all tracks in the region of interest. This estimator cannot be used when employing the delta-tracking method because the sampled neutron paths may extend over several material regions and the discontinuity points are not known [8].

An alternative to the track-length flux estimator is the collision flux estimator:

$$R_{\text{cfe}} = \sum_{i=1}^{I} \frac{f_i}{\Sigma_{\text{tot,i}}}, \tag{2.6}$$

where Σ_{tot} is the material total macroscopic cross section at the collision site and the summation is over all collisions within the region of interest [8]. This flux estimator is problematic in that it often results in poor efficiency for tallies scored in regions of low volume or with low collision density. However, the use of the collision flux estimator should not be considered a disadvantage for the implementation of the delta-tracking method in WARP; this estimator is already the code's flux tally method [1].

In WARP, the flux tally kernel scores collisions in cell j with volume V_j as shown in Equation 2.7, where $\Sigma_{\text{tot,j}}(E)$ is the total macroscopic cross section at energy $E_i < E < E_{i+1}$ and $N_{i,j}$ is the number of collisions in that volume [1]:

$$\bar{\phi}_{i,j} = \frac{1}{V_j} \sum_{i=1}^{N_{i,j}} \frac{1}{\Sigma_{\text{tot,j}}(E)} \,. \tag{2.7}$$

2.2 Ray tracing in WARP

WARP uses the NVIDIA OptiX ray tracing framework for geometry representation [1]. OptiX is a highly-optimized, scalable framework developed by NVIDIA for building ray tracing-based applications. It allows for user-written applications that use ray tracing for graphics, collision detection, and other fields of interest [9]; the flexibility of the framework makes it suitable for this application in Monte Carlo neutron transport. The OptiX engine is composed of a host-based API that defines ray tracing based data structures, which works in combination with a CUDA C-based programming system that can produce new rays, intersect rays with surfaces, and respond to those intersections [9].

In WARP and other Monte Carlo codes that use ray tracing to track particle locations, material information is updated when a sampled interaction distance is greater than the nearest surface distance. The neutron is moved to the boundary that it will cross, the material information is updated to the material that the neutron is entering, and the interaction distance is resampled in the new material using the same direction of flight [1]. WARP uses an algorithm that uses the OptiX framework to perform the ray tracing required to determine the entering material. Since all of the system geometric information is stored in the OptiX context [1], this allows the material information update to be done readily.

WARP represents geometric volumes as the intersection of the space inside of a given cell with the space outside any cells that are nested inside of it [1]. For example, if two objects were designated to be centered at the same coordinate point with the inner object entirely encompassed by the outer object, the space between the two objects would belong to the outer object while the volume inside of the inner object would all belong to the inner object. This is important to note because it sets the type of algorithms that can be used for geometry handling.

WARP's material query algorithm is shown in Figure 2.1 [1]. A list of surface intersections, ordered by proximity, is generated by iteratively ray tracing along a neutron's direction of flight and adding the closest surface number to the hit list each time. Tracing is stopped once the ray hits a predefined outer cell that contains all other cells [1]. Because all surfaces are closed, the ray intersects any cell surface twice if its origin is not nested in the cell in which the ray originated. When the list of surface intersections is created, the double entries are removed, resulting in a list of cells in which the neutron is nested. The first entry of the list is the cell in which the neutron is located at the time of the material query and the second entry of the list is the cell that the neutron is entering [1].

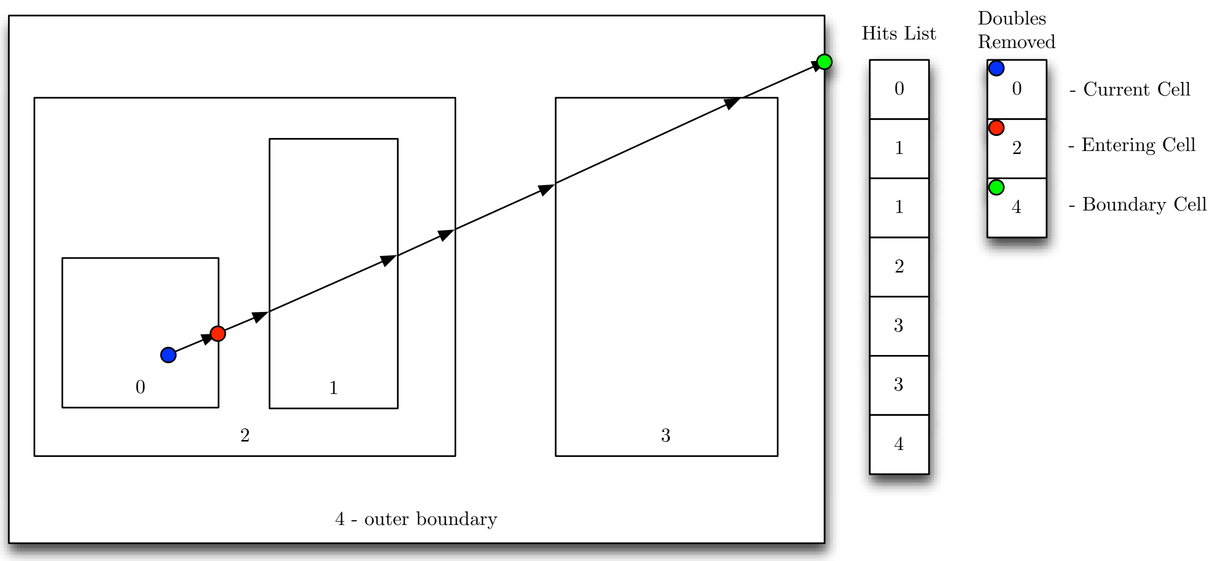

Figure 2.1: The point-in-polygon-like algorithm for determining the entering cell/material number by using ray tracing.

One particular issue with ray tracing is that cell descriptions are mathematically exact, but they are represented by inexact numbers. Floating-point roundoff can cause a neutron to be placed slightly behind a boundary instead of at the actual boundary. This situation can largely be prevented by using an OptiX scene "epsilon", which designates the minimum distance away from the source point at which intersections are allowed to occur [1]. The scene epsilon helps ensure that a trace starts after a boundary such that accurate results are calculated. For the algorithm to be used effectively, it is important that the scene epsilon is set to an appropriate value for the problem geometry [1].

The scene epsilon, however, can also cause inaccurate material queries in certain geometry configurations. If the thickness of an intersected object is less than the scene epsilon, the material query algorithm may only count a single intersection of the object instead of two. The code will then assign the neutron the thin object's material rather than determining that the neutron moved through the thin material and into the subsequent one [1]. In many cases, this can be avoided by performing the surface intersection in the direction of the neutron flight path and then performing the material query in the z-direction only. At this point in development, the objects in WARP are all z-aligned, optically thick shapes. Thus, the previously described incorrect material query will not happen if the material query is done in the z-direction because the ray will only intersect planes perpendicular to it [1].

An additional issue that can occur in using this algorithm happens when cells have coincident surfaces. In this case, the number of the cell that is actually intersected is undefined and the following trace iteration will skip the intersection of the coincident cell. One way to avoid this would be to ensure that coincident surfaces are more than one scene epsilon apart [1]. In some cases, the neutron mean free path is significantly larger than this introduced gap and this approximation will not affect the results. However, if the mean free path is smaller than the gap (which may happen in cells next to strong absorbers), errors could be introduced by the approximation. An automatic way of determining this may be an area of future development in WARP [1].

2.3 Delta-tracking in existing Monte Carlo neutron transport codes

Over the years, many codes have had delta-tracking available as at least an option for neutron tracking. We will point out, however, that none of these codes were implemented on GPUs.

The concept of delta-tracking was introduced in the "HOLE" routines in the GEM code, which was designed to study criticality safety problems in chemical plants with the primary objective of using nuclear data in the finest detail as was possible [10]. Subsequent developments in the code enabled calculations that included all significant structural features of a complete reactor.

Geometric representation in GEM was similar to that in WARP, discussed above. The system was divided into regions by boundaries that form closed, nested surfaces that can touch or coincide but cannot intersect. Regions filled by a single homogeneous material were specified only by their atomic composition and density, while the geometry of a heterogeneous region was handled with the delta-tracking method in the HOLE routine [10]. This method was used to reduce the mean free path of all materials in the region to the same value such that they were independent of internal boundaries.

RCP01, a Monte Carlo program used to analyze geometries of interest in nuclear design and analysis of LWRs and their associated technologies, used delta-tracking as an optional acceleration method [11]. “Fast” tracking in subcell regions could be turned on by the user; this switched on delta-tracking based upon the maximum cross section in the subcell. This method could not be used in conjunction with point detectors.

Delta-tracking was also an optional feature in RACER, a system of codes used for Monte Carlo nuclear reactor physics analysis [12]. The “Monte Carlo - Vectorized” (MCV) module offered the option of using the delta-tracking method on the basis that it is potentially more efficient than “regular” tracking in some instances. The user needed to specify the tracking scheme for each neutron tally group, resulting in the overall use of a mixed tracking scheme that varied with neutron energy. This was necessary for both consistent tally estimation and efficient neutron tracking.

The “MCU-PD” version of the Monte Carlo Universal code uses the option of delta-tracking for geometry simplification [13]. Systems in MCU-PD are represented as unions of homogeneous geometric areas, each described as a boolean combination of a set of simple bodies (that is, the code uses the method of combinatorial geometry). One limitation of the method of combinatorial geometry is that complex boundary surfaces must be approximated by a large number of zones. In MCU-PD, use of the delta-tracking method removes this limitation.

The VMONT code is a fuel assembly analysis code that uses a vectorized Monte Carlo method for neutron transport calculation and incorporates delta-tracking to decrease code runtime and processing [14]. It was found in this code that without delta-tracking, neutron flight path analysis took up about 80% of computing time; implementation of delta-tracking resulted in a speedup factor of 1.5 relative to this figure.

MONK, a Monte Carlo neutronics code for the solution of criticality safety and reactor physics problems, has its origins in the GEM code discussed above. Along with the Monte Carlo code MCBEND, MONK is part of the ANSWERS codes suite, which is used for calculations involving a variety of reactor types, including experimental reactors [15]. Both codes use a common geometry package containing the component “Hole Geometry”, which uses delta-tracking. Hole Geometry is complementary to the conventional ray tracing “Fractal Geometry”; the Hole Geometry package facilitates exact modeling of complicated geometry configurations that would be impossible to model using the simple bodies in the Fractal Geometry package.

Delta-tracking is an option in the MORET 5 code, a Monte Carlo code designed to perform calculations to support criticality safety assessments [16]. MORET 5 uses a combinatorial geometry scheme and allows use of delta-tracking to reduce runtime in scenarios involving complex geometries, especially those with complex shapes or that have a large number of volumes with dimensions smaller than the neutron mean free path (such as a pebble bed reactor).

The Serpent Monte Carlo reactor physics burnup calculation code employs a unique combination of both ray tracing and delta-tracking in its geometry routine. The code originally used only the delta-tracking method, which was chosen for the sake of simplicity, but efficiency problems induced by the presence of heavy localized absorbers when using the delta-tracking method led to the hybrid geometry method currently used in Serpent [17].

Serpent switches to using ray tracing when collision sampling efficiency is low. Selection between the two methods is done by comparing the neutron mean free path resulting from the majorant to the physical value of the mean free path given by the material total cross section at the neutron's location [8]. If

$$\frac{\ell_{\mathrm{maj}}(E)}{\ell_m(E)} = \frac{\Sigma_{\mathrm{tot,m}}(E)}{\Sigma_{\mathrm{maj}}(E)} > 1 - c, \tag{2.8}$$

where ℓ_m is the material-calculated path length and ℓ_{maj} is the majorant-calculated path length, the neutron path length is sampled using the majorant cross section and rejection sampling is subsequently carried out at the collision point. The constant c is the predefined cutoff criterion valued between 0 (no delta-tracking) and 1 (no ray tracing). After several parametric studies were performed to assess the runtime of the Serpent code for different reactor geometries with varied values of c, the cutoff criterion value was set to a default fixed value of 0.90 [8] but can be changed by the user.

The Serpent parametric studies also compared the flux estimates in two of the LWR lattice configurations to demonstrate that the decrease in runtime from the use of the delta-tracking method is not outweighed by poor statistics. Comparisons were done with the figure-of-merit (FOM) metric, defined below, which incorporates both the calculation time T and the relative statistical error $\Delta x/x$ of the result [8].

$$\mathrm{FOM} = \frac{1}{T(\Delta x/x)^2} \tag{2.9}$$

It was found that, for parameters integrated over the entire geometry, the accuracy of the two methods is comparable.

2.4 Vector computing as a basis for GPU algorithims

The origin of the ideas in WARP is research done in the 1980s and 1990s for mapping the Monte Carlo and collision probability methods onto vector computers [1]. These methods on those machines used an "event-based" algorithm in which neutrons are organized and processed according to their required operation. If a neutron is about to undergo inelastic scattering, its data are put into the inelastic scattering buffer. The same process is done for all reaction types: neutrons inducing fission are placed in the fission buffer, and so on. Vector processing calculations such as these are more generally referred to as "single instruction multiple data", or SIMD, processes. Since GPUs are massively parallel and rely on SIMD, the WARP code uses a modified event-based algorithm for GPU-accelerated Monte Carlo neutron transport.

The RACER and VMONT codes were both vectorized Monte Carlo codes that used delta-tracking for the purpose of reducing runtime and processing. It follows logically that, since delta-tracking has been shown to be efficient in these vector machine codes, it has the potential to be efficient when used on a GPU-accelerated, vectorized Monte Carlo code such as WARP.

Chapter 3

Implementation

3.1 Delta-tracking in WARP

The implementation of delta-tracking in WARP is somewhat similar to the implementation in RACER and Serpent in that it uses a combination of ray tracing and delta-tracking. To understand what is happening in WARP, what follows is an explanation of the general delta-tracking algorithm implemented, a comparison to ray tracing, discussion of the specific implementation in WARP, and finally some comments about the performance potential of using delta-tracking on GPUs for the first time.

In WARP, the delta tracking algorithm executes these steps:

- get current neutron positions using OptiX trace;
- calculate majorant cross section, sample path lengths, update neutron coordinates;
- update neutron positions and material information using OptiX trace;
- calculate material total cross section and perform rejection sampling;
- determine neutron interaction.

For direct comparison, tracking particles via ray tracing only in WARP is executed as:

- get current neutron positions using OptiX trace;
- calculate macroscopic total cross section and sample path length;
- determine with which isotope neutron interacts;
- move neutron to collision site or cell boundary, whichever is closer;
- determine neutron interaction.

From there, with either tracking scheme, neutrons are grouped based on the interactions that they are about to undergo and the vectors are subsequently processed.

3.2 Implementation

In WARP and other Monte Carlo neutron transport codes, particle positions are updated as:

$$\begin{aligned} x &\mathrel{+}= \ell\hat{x}, \\ y &\mathrel{+}= \ell\hat{y}, \\ z &\mathrel{+}= \ell\hat{z}, \end{aligned} \tag{3.1}$$

where ℓ, the sampled travel distance is defined as:

$$\ell = \begin{cases} \frac{-\log\xi}{\Sigma_{\text{tot}}} & \text{with ray tracing,} \\ \frac{-\log\xi}{\Sigma_{\text{maj}}} & \text{with delta-tracking.} \end{cases} \tag{3.2}$$

If ray tracing is being used, a check is performed between the sampled neutron flight distance and the distance to the nearest boundary crossing on that trajectory. If the boundary crossing is closer than the sampled distance, the particle position is instead updated as

$$\begin{aligned} x &\mathrel{+}= s\hat{x} + \varepsilon x_{\text{norm}}, \\ y &\mathrel{+}= s\hat{y} + \varepsilon y_{\text{norm}}, \\ z &\mathrel{+}= s\hat{z} + \varepsilon z_{\text{norm}}, \end{aligned} \tag{3.3}$$

where s is the distance to the surface to be crossed. The epsilon term added to the position is required by the use of the OptiX framework; the inclusion of this term ensures that the neutron is sufficiently far into the new cell such that it can be detected by OptiX and the corresponding material updated accordingly. The second term uses vector norms rather than the vector directions in Equation 3.2 to move particles away from the surface in a perpendicular manner rather than potentially along a surface boundary (which could cause issues with OptiX detection). In this case of a particle crossing a boundary with ray tracing, the particle is stopped at this updated location and the flight path distance must be resampled using the total macrscopic cross section of the new material.

Although delta-tracking circumvents the above resampling in the case of surface crossings, there is a potential need to resample at each collision site. The collision site is accepted as an actual reaction with probability

$$P_{\text{col}}(E) = \frac{\Sigma_{\text{tot,col}}(E)}{\Sigma_{\text{maj}}(E)} \tag{3.4}$$

and rejected as a "virtual" collision otherwise. It is important to note that this rejection sampling must use the total macrscopic cross section of the material in which the collision is being considered and not the material in which the neutron was originally located. If the collision is considered virtual, a new path length is sampled with the majorant cross section again, and the transport process continues.

It should be noted that calculating the majorant cross section in WARP is done naïvely:

$$\Sigma_{\mathrm{maj}}(E) = \max\{\Sigma_{\mathrm{tot},1}(E), \Sigma_{\mathrm{tot},2}(E), \ldots, \Sigma_{\mathrm{tot},\mathrm{M}}(E)\} \,, \tag{3.5}$$

where M is the number of materials in the entire system. One potential area of development would be to calculate the majorant cross section along the neutron's trajectory (as specified in Equation 2.3) rather than this "global" majorant cross section.

With either tracking method, once the neutron has been determined to undergo an actual collision the reaction it will experience is chosen using the microscopic cross sections of the material in which the particle is located. Following this, the reaction is processed, and transport continues as above.

3.3 Performance potential of delta-tracking in WARP

Because the OptiX trace must be done twice in each instance of the delta-tracking process and the origina ray tracing method only necessitates one trace, it may be that the current implementation of delta-tracking may not result be faster than standard ray tracing for simple geometry and material configurations. It has been seen in various other Monte Carlo neutron transport codes that the bulk of the runtime is involved with ray tracing, and WARP is not too different from other codes in that regard. This issue could potentially be circumvented if a different framework were to be used rather than OptiX, but this would require a sizable overhaul of the code and was not considered for this project. Further, the OptiX framework is optimized for NVIDIA GPUs and at this time it is unclear what a new, better choice might be.

However, it is expected that, like in other codes, delta-tracking may be more efficient in WARP when considering certain geometry configurations. Like Serpent, WARP is intended for calculations in nuclear reactor analysis. Many reactor configurations consist of lattices composed of many geometry and material boundaries, causing simulations that use ray tracing to run slowly because neutron flights stop at each boundary crossing. Thus, the time saved in not stopping neutron flights at boundary crossings may compensate for having to call an additional OptiX trace in the delta-tracking method implemented in WARP.

Chapter 4

Results

The question that this research is answering is whether this expected behavior of delta-tracking, which is to perform within statistical accuracy to ray tracing with lower runtime, holds on GPUs compared to CPUs. The expected behavior is of interest because it is crucial to the study that results are statisically accurate; runtime is irrelevant if the calculations are performed incorrectly. Additionally, it is of interest to developers to see if delta-tracking is faster than ray tracing on an advanced computer architecture.

The problems investigated here are designed to answer these questions by testing a variety of problems relevant to reactor physics, ranging from very simple systems comprised of one cell and one isotope to a complex configuration of hundreds of cells and multiple isotopes. This strategy allows detailed investigation of the algorithm as well as demonstrating behavior for problems of interest.

Here, the version of WARP with delta-tracking is compared against the version of WARP with ray tracing. The original ray tracing version of the code has been benchmarked against both MCNP and Serpent [18] so, for this work, it is sufficient to assess how well results from the delta-tracking version of WARP match those generated by the ray tracing version. The success criteria of the delta-tracking version of WARP will be whether the results are statistically accurate compared to the ray tracing version of WARP and if the calculation time is lower than the ray tracing version of WARP.

As WARP is still in development, it currently only supports four geometry and material configurations. These configurations can be lumped into two groups: homogeneous systems and heterogeneous systems. The geometry parameters for the four configurations are listed in Table 4.1 [1]. The pin cell and assembly cases contain water at 3 g/cm^3, which was an error introduced in the original calculations [18]. We have retained this water density so we can make direct comparisons. Although this is clearly unphysical, it does not have an impact on the simulation results and, since both versions of WARP contain the same geometry configurations, the comparisons are consistent.

All calculations were performed on an NVIDIA GeForce GTX TITAN Black graphics card. Additionally, the following results all use these particular specifications:

Table 4.1: Geometry and materials used in the test cases.

Test	**Number and Type of Cells**	**Material(s)**	**Isotope(s)**	**Density**
Jezebel	1 sphere; r = 5.1cm	Fuel	(1.00) ^{239}Pu	19.816 g/cm^3
Homogenized Block	1 cube; r = 10cm	Hom. Fuel	(0.90) ^{238}U (0.10) ^{235}U (3.00) ^{16}O (2.00) ^{1}H	10 g/cm^3
Pin Cell	1 cuboid, 10x10x50cm 1 cylinder; r = 1cm, z = 40cm	Fuel	(0.90) ^{238}U (0.10) ^{235}U (2.00) ^{16}O	15 g/cm^3
		Water	(1.00) ^{16}O (2.00) ^{1}H	3 g/cm^3
Hex Assembly	1 cube, 84cm^3 631 cylinders; r = 1cm, z = 40cm	Fuel	(0.90) ^{238}U (0.10) ^{235}U (2.00) ^{16}O	15 g/cm^3
		Water	(1.00) ^{16}O (2.00) ^{1}H	3 g/cm^3

- 500 000 neutron histories,
- 20 inactive cycles followed by 40 active cycles,
- vacuum boundary conditions, and
- Serpent ENDF/B-VII ACE data at 300K.

For each of the configurations, it is expected that the delta-tracking version of WARP will arrive at a statistically accurate calculation of the system's effective multiplication factor and that the neutron flux spectra generated by both versions of the code will be consistent. Additionally, it is expected that, for calculations involving homogeneous systems, the delta-tracking version of WARP will be slower than those performed with the ray tracing version of WARP. This is because, compared to the ray tracing routine, the delta-tracking routine requires an additional OptiX trace to update neutron locations after the particles have been moved. This is done in order to retrieve the updated material information required for the rejection sampling incurred in delta-tracking. We conduct homogeneous system comparisons in order to verify algorithm correctness. For the heterogeneous systems, it is expected that the delta-tracking version of WARP may be faster than the ray tracing version; not stopping the neutrons at each boundary crossing may save more time than is induced by the extra neutron position updates.

4.1 Homogeneous systems

The delta-tracking method was first tested in the two homogeneous systems to ensure that the method delivers accurate results compared to those in the original version of the code. It was expected that the calculated results would be accurate within statistical error but that the calculations performed using delta-tracking would be slower than those performed with ray tracing. Because there is only one cell and therefore one boundary in these systems, delta-tracking should not have any advantage over ray tracing as ray tracing is still used to check whether or not a neutron has leaked out of the system. Additionally, the delta-tracking method requires the code to do an extra update of neutron position when compared to the ray tracing algorithm, and it is expected that these supplementary calculations will incur more runtime.

Bare plutonium sphere

The "Jezebel" test is a bare plutonium sphere and is considered to be a standard criticality test [19]. The actual geometry configuration is a sphere of plutonium-239 with a radius of 5.1 cm. Computational results for the Jezebel scenario are shown in Table 4.2 and Figure 4.1.

Table 4.2: Calculated multiplication factors and runtimes for the Jezebel configuration.

Method	$\mathbf{k_{eff}}$	**Runtime**
Ray tracing	$1.027853 \pm 3.8304 \times 10^{-4}$	16.17 s
Delta-tracking	$1.026794 \pm 3.5830 \times 10^{-4}$	22.73 s

The multiplication factors calculated by each method match within statistical uncertainty and the flux spectra are nearly (if not actually) identical. The runtimes are comparable for both methods. Here, it should not be expected that the delta-tracking method would be considerably different compared to the ray tracing algorithm, as the system only contains one cell with one material composed of a single isotope.

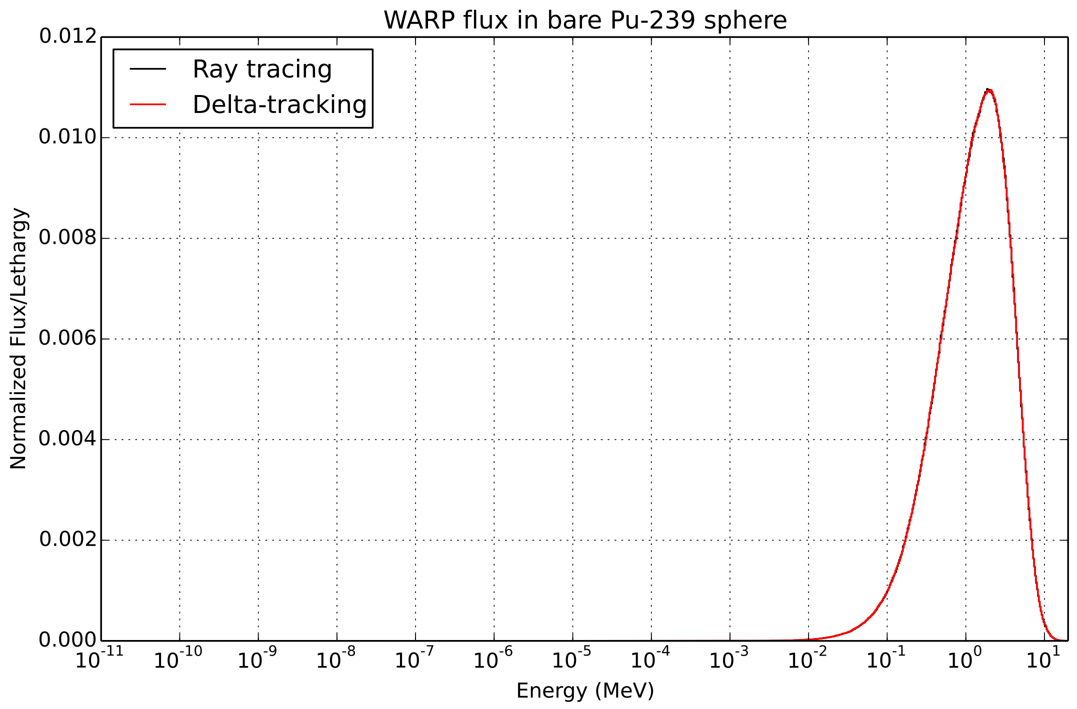

Figure 4.1: Normalized flux spectra for the Jezebel configuration generated by both versions of WARP.

Homogenized block

The "homogenized block" test consists of one cube with edges 20 cm in length. The cube is composed of a homogeneous mixture of 10% ^{235}U enriched UO_2 and water at a 1:1 ratio [1]. While still a homogeneous case, the homogeneous block is more complex than the Jezebel scenario in that its single material is composed of multiple isotopes. We are again testing that the delta-tracking method calculates accurate results, but it is expected that the extra calculations involved in determining the majorant cross section and the extra OptiX trace needed in the delta-tracking routine will cause an increase in runtime. Results for the homogenized block configuration are shown in Table 4.3 and Figure 4.2.

Table 4.3: Calculated multiplication factors and runtimes for the homogenized block configuration.

Method	**k_{eff}**	**Runtime**
Ray tracing	$1.213833 \pm 2.9255 \times 10^{-4}$	28.14 s
Delta-tracking	$1.213983 \pm 2.1413 \times 10^{-4}$	30.51 s

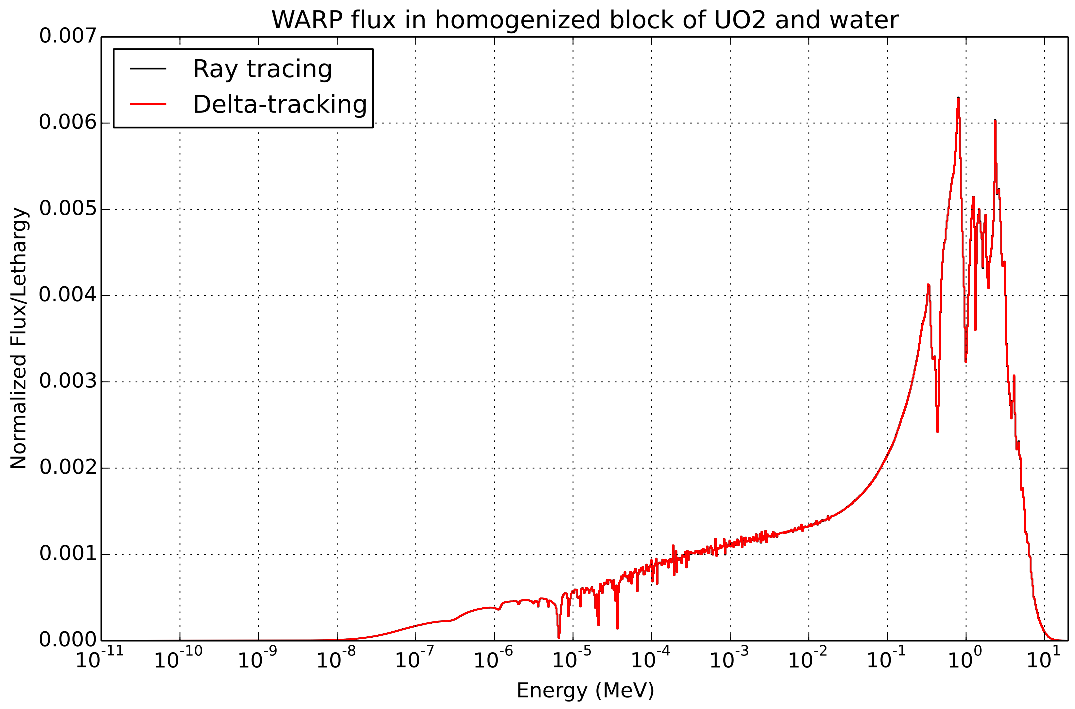

Figure 4.2: Normalized flux spectra for the homogenized block configuration generated by both versions of WARP.

Again, the multiplication factor calculated by the delta-tracking version of WARP matches that calculated by the ray tracing version of WARP (within statistical error). The flux spectra are also consistent with one another. However, this case shows a greater discrepancy in the runtimes of each of the two methods. The version of the code that uses the delta-tracking method is considerably slower, likely because of the extra calculations required to compute the majorant cross section and the additional OptiX trace required in each run.

4.2 Heterogeneous systems

For the heterogeneous systems, we are once again testing that the delta-tracking method produces results that are statistically accurate compared to results calculated using the ray tracing algorithm. However, for these systems it is expected that the delta-tracking method may faster than the ray tracing algorithm. Although additional calculations are incurred in the determination of the majorant cross section and the additional OptiX trace required for the delta-tracking routine, the time saved in not having to stop neutrons at each material boundary may be enough to result in a reduction in runtime with the use of the delta-tracking method.

Fuel pin cell

The "pin cell" test contains a bare UO_2 cylinder encased in a block of water. The pin has a diameter of 1 cm and a length of 40 cm; the dimensions of the surrounding water block are $10 \times 10 \times 50$ cm [1]. This test has two materials, each with multiple isotopes, and two cells. Results can be seen in Table 4.4 and Figures 4.3 and 4.4.

Table 4.4: Calculated multiplication factors and runtimes for the pin cell configuration.

Method	$\mathbf{k_{eff}}$	**Runtime**
Ray tracing	$3.798339 \times 10^{-1} \pm 6.3999 \times 10^{-4}$	3.66683 min
Delta-tracking	$3.822616 \times 10^{-1} \pm 4.9758 \times 10^{-4}$	3.703 min

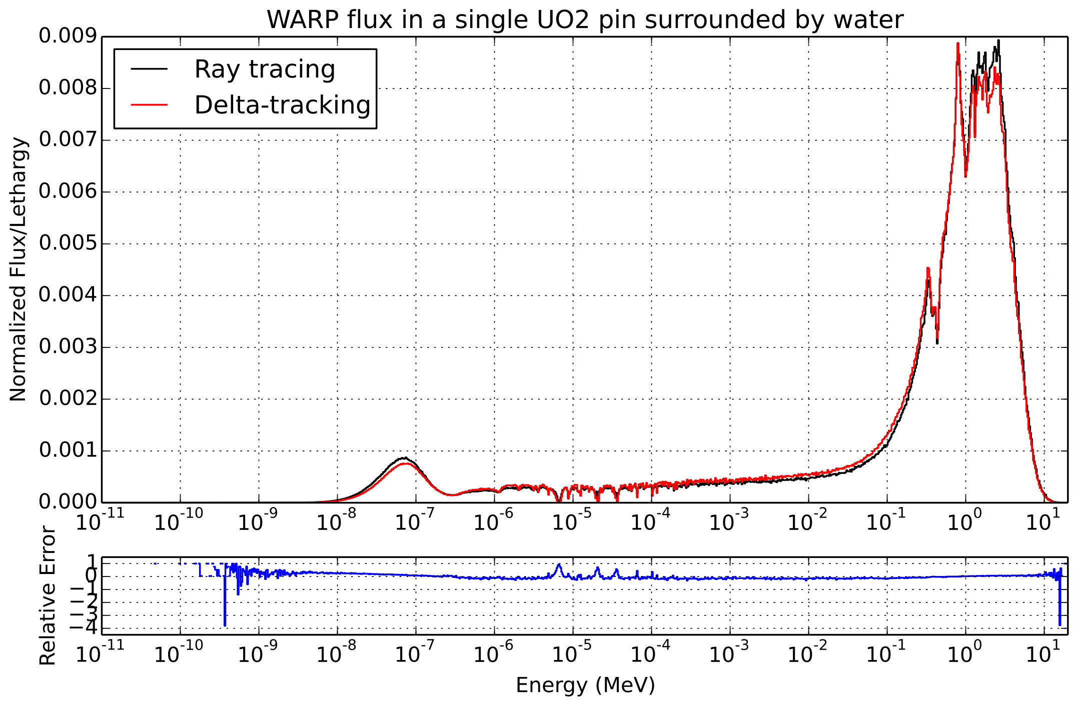

Figure 4.3: Normalized flux spectra for the UO_2 **cylinder** in the pin cell configuration generated by both versions of WARP.

In this basic, but heterogeneous, configuration, delta-tracking does not accurately calculate the multiplication factor for the system. Although the two values look similar, they do not match within statistical uncertainty. Additionally, large discrepancies are seen in the neutron flux spectra in both the fuel pin as well as the water surrounding it.

Figure 4.3 shows that the flux spectrum in the fuel generated by the delta-tracking version of WARP has slight but noticeable discrepancies compared to the flux spectrum

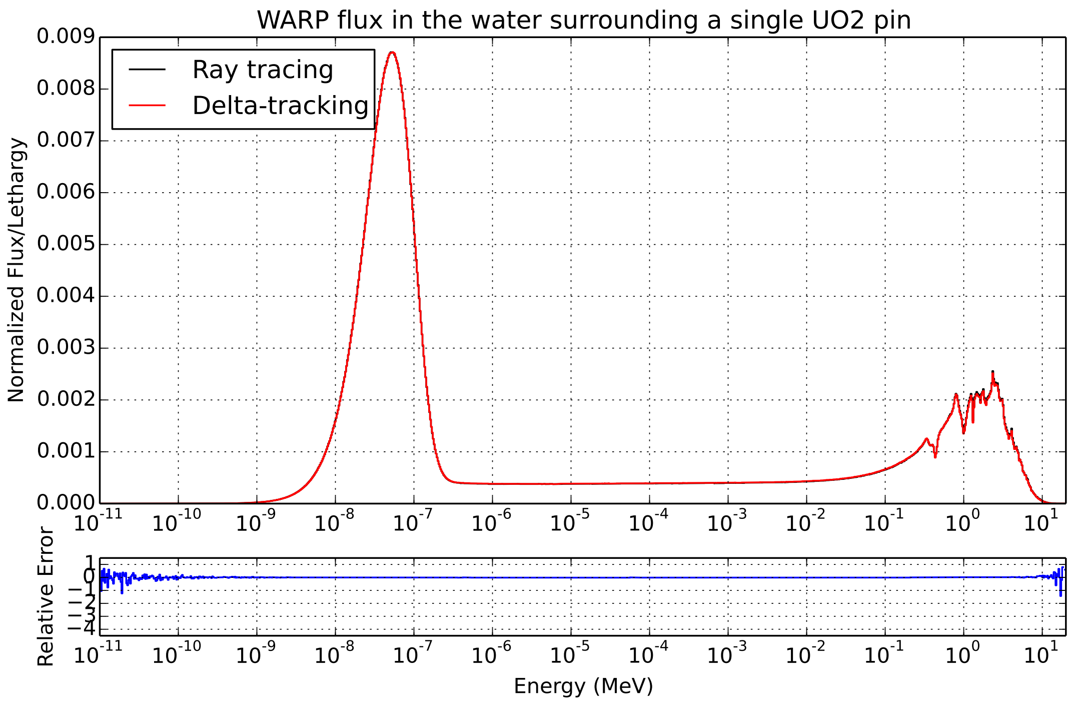

Figure 4.4: Normalized flux spectra for the **water** block in the pin cell configuration generated by both versions of WARP.

generated by the original ray tracing version of the code. The fast and thermal flux are both underestimated, while the epithermal flux is overestimated. However, Figure 4.4 shows that the flux spectrum in the water generated by the delta-tracking version matches the spectrum generated by ray tracing quite well.

Note that the issues in the calculation of the effective multiplication factor and the normalized flux spectrum in the fuel of the pin cell align. That is to say, there are more neutrons counted in the fuel in the case of delta-tracking, and the overestimated value of k_{eff} also indicates greater numbers of neutrons than should actually be present. Since the spectral issue is only seen in the fuel (and the value of k_{eff} applies to the entire system), we may project that the original code perhaps has a bug in how fission events are handled. Since no fissions occur in the water, this could be a potential source of the error seen above.

In this case, delta-tracking is again slower than ray tracing. The extra calculation time likely comes from the calculation of the majorant at every instance of path length sampling as well as resampling incurred by the use of the majorant in a system with multiple materials.

Fuel pin assembly

The "fuel pin assembly" test consists of 631 UO_2 cylinders arranged in a hexagonal lattice surrounded by a cube of water with edges of length 84 cm. The material compositions and cylinder dimensions are identical to those described above in the pin cell case, but the large number of objects in this scenario is intended to highlight the effect of introducing many geometric objects into a problem [1]. Computational results for the fuel pin assembly configuration are shown in Table 4.5 and Figures 4.5 and 4.6.

Table 4.5: Calculated multiplication factors and runtimes for the fuel pin assembly configuration.

Method	**k_{eff}**	**Runtime**
Ray tracing	$1.445201 \pm 2.3456 \times 10^{-4}$	4.36917 min
Delta-tracking	$1.415779 \pm 1.9934 \times 10^{-4}$	4.57683 min

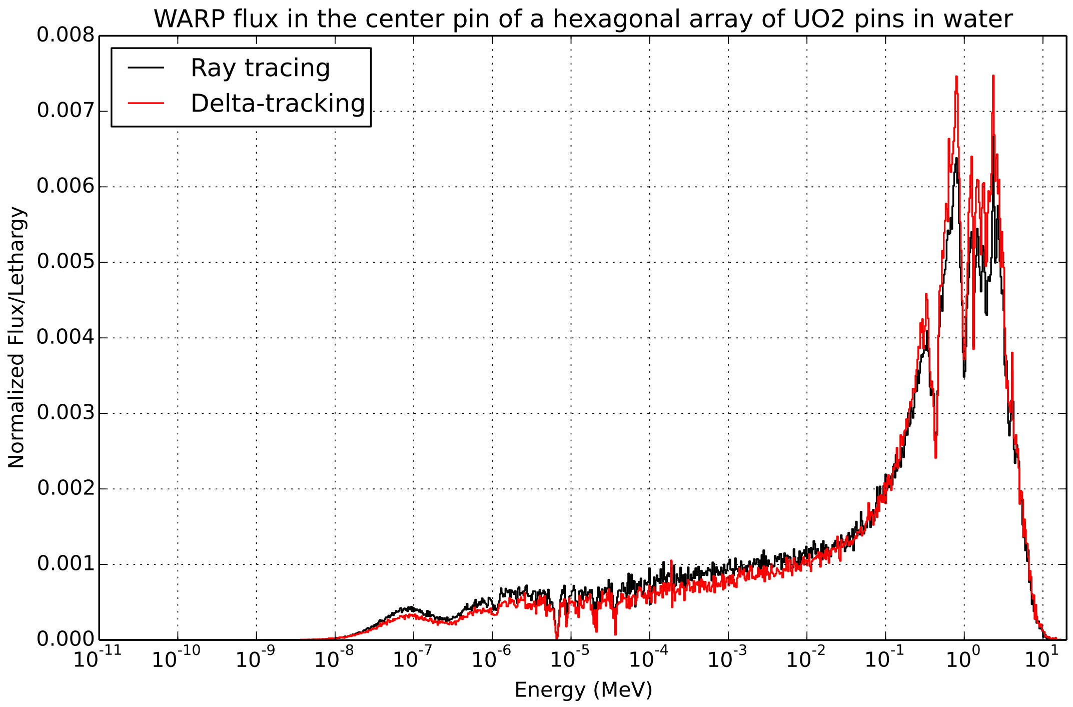

Figure 4.5: Normalized flux spectra for the **center pin** of a hexagonal array of UO_2 pins in water generated by both versions of WARP.

The results for the fuel pin assembly simulation suffer from the same geometry issues discussed earlier in the pin cell simulation results. Using the delta-tracking method causes the neutron multiplication factor to be grossly underestimated. Spectral issues similar to

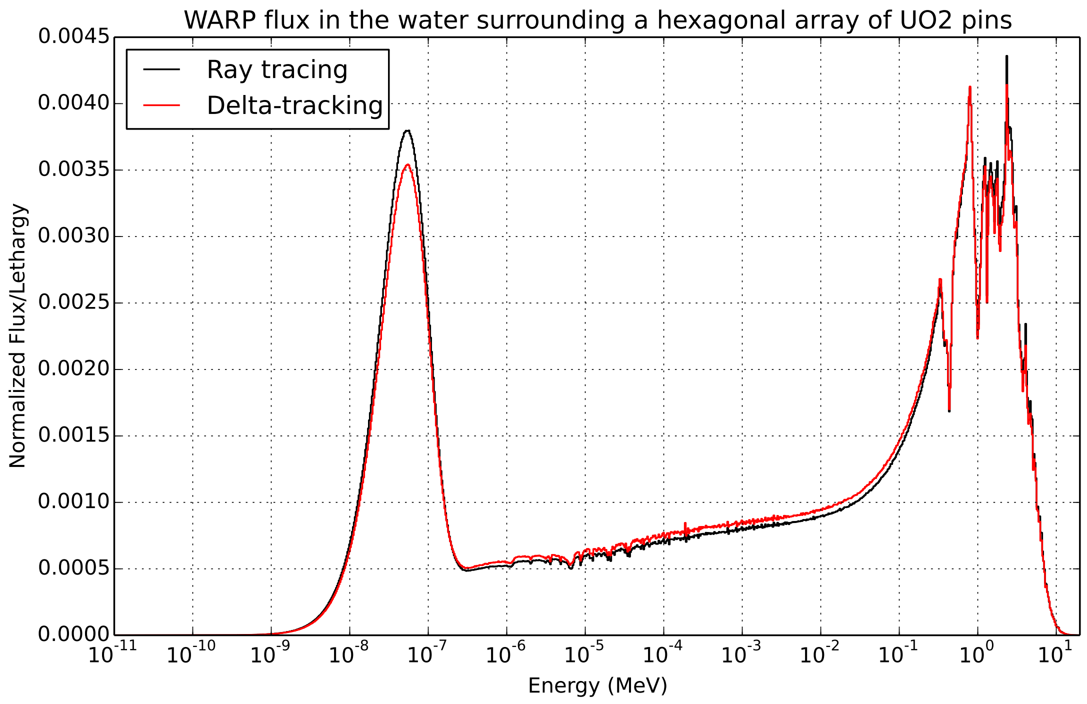

Figure 4.6: Normalized flux spectra for the **water** surrounding a hexagonal array of UO_2 pins generated by both versions of WARP.

those in the pin cell configuration are observed; compared to the spectra generated using ray tracing, too many fast neutrons are in the center fuel pin in the assembly and too few thermal neutrons are in the water surrounding the fuel pins. Given the mismatch in the single pin, this behavior is consistent with expectation.

In addition to these geometrical errors, the delta-tracking method is again slower than the ray tracing algorithm. Computing the majorant cross section and resampling neutron interactions requires additional runtime on top of the remainder of the Monte Carlo algorithm.

4.3 Experimental homogeneous systems

The discrepancies between the two methods in the results of the heterogenous systems prompted the question of whether the incorrect behavior was caused by the implementation of the delta-tracking method or the original code as a whole. The homogeneous systems discussed above were not sufficient to answer this question as they contained only one outer boundary.

Taking these notions into consideration, a homogeneous system with artificial boundaries, shown below in Figure 4.7, was created to test the delta-tracking method against the ray tracing method once more. The material and geometry configuration is identical to the Jezebel configuration described in Table 4.1 but with an artifical boundary inserted in the sphere. A more extreme version of this configuration has four artificial boundaries and is shown in Figure 4.8.

The configurations are obviously non-physical; they were contrived explicitly for the purpose of testing the delta-tracking method in an environment more complex than the basic homogeneous systems in Section 4.1 but less complex than the full-blown heterogeneous configurations in Section 4.2. From the results of these simulations we may infer that if the delta-tracking results match those calculated with ray tracing, the issues observed in Section 4.2 are issues with the code that already existed prior to the introduction of the delta-tracking method. If the results are inconsistent, however, we may conclude that there is an error in the implementation of the delta-tracking method.

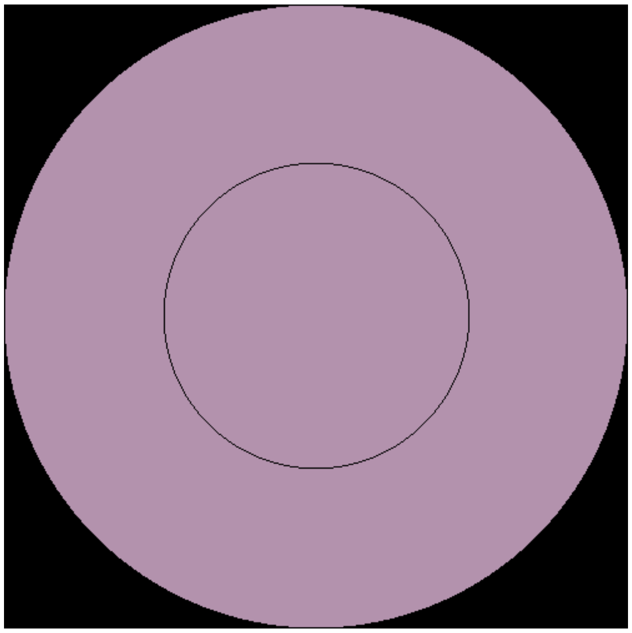
Figure 4.7: Plot of “Jezebel” configuration with an inserted artificial boundary.

Results of simulations of the two experimental homogeneous systems follow in Tables 4.6 and 4.7. The results show that in each case, the effective multiplication factor calculated using the delta-tracking method corresponds to that calculated using the ray tracing method (within statistical error). It should be noted that the ray tracing method referred to here is not the algorithm discussed in Section 2.2 but the algorithm discussed in Appendix A. In testing these experimental homogeneous configurations, a major error was found in the original ray tracing algorithm and it was replaced with a more correct method.

Again, the runtimes of the two methods are comparable, with the delta-tracking method actually outpacing the new ray tracing algorithm for the case of the Jezebel configuration with four artificial boundaries. Because the delta-tracking method is consistent with ray

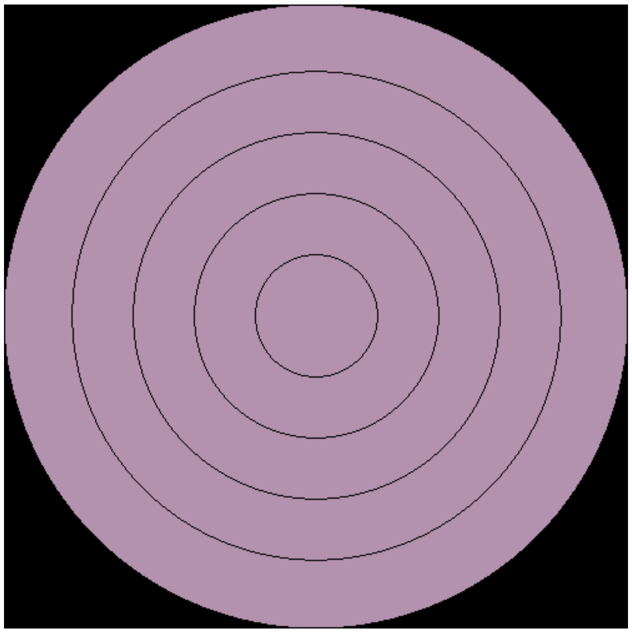

Figure 4.8: Plot of "Jezebel" configuration with four artificial boundaries.

Table 4.6: Calculated multiplication factors and runtimes for the Jezebel configuration with one artificial boundary.

Method	$\mathbf{k}_{\text{eff}}$	**Runtime**
Ray tracing*	$1.027356 \pm 3.4527 \times 10^{-4}$	20.29 s
Delta-tracking	$1.027218 \pm 2.7643 \times 10^{-4}$	23.79 s

Table 4.7: Calculated multiplication factors and runtimes for the Jezebel configuration with four artificial boundaries.

Method	$\mathbf{k}_{\text{eff}}$	**Runtime**
Ray tracing*	$1.027238 \pm 2.7643 \times 10^{-4}$	24.27 s
Delta-tracking	$1.027425 \pm 2.7085 \times 10^{-4}$	24.06 s

tracing for the cases of configurations with a single material and multiple cells, we may infer that the issues seen in Section 4.2 are due to exacerbation of preexisting bug(s) in the code by the delta-tracking method. As a result, one cannot determine whether or not delta-tracking is a viable neutron tracking routine for good performance on GPUs. However, the performance of the routine observed in the experimental homogeneous cases above indicates the potential of the delta-tracking method on GPUs.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this work, delta-tracking was implemented as a neutron tracking routine in a GPU-accelerated Monte Carlo neutron transport code and the results were compared for accuracy and speed with respect to the original ray tracing version of the code. For systems comprised solely of a homogeneous material, the delta-tracking method produced results consistent to within statistical uncertainty of those calculated using ray tracing. The delta-tracking method failed to achieve consistent results when simulating heterogeneous systems, however. Further testing of homogeneous systems with artificial boundaries in place revealed that those boundaries do not skew the results of the delta-tracking routine such that they are outside of the level expected. Thus, it may be concluded that there are existing bugs in the original ray tracing version of the code that the delta-tracking routine exacerbates, causing results from the latter method to be incorrect.

The speeds of the two physics methods were comparable, with ray tracing often slightly faster than delta-tracking. The relative lag of the delta-tracking method can be easily explained by the fact that it requires an additional OptiX trace in addition to the single trace required by ray tracing. However, since the runtimes of both methods were fairly close we assert that, for configurations with many cells, the homogenization of the delta-tracking may mitigate the effect of the slowdown from the second trace, allowing it to perform better than ray tracing.

However, before further scaling studies can be done with the delta-tracking method, the original core of the code must be closely examined to resolve existing bugs. After the prevalent issues have been solved, delta-tracking may be revisited.

5.2 Future Work

The first point of future work is to resolve the issues in the existing ray tracing code. This will be done in several thrusts. Work is planned to reorganize the main data structures in the code such that they are simpler, more intuitive, and more easily understood by interested developers new to the code. Additionally, more unit testing in the code will provide useful information about the functionality of the individual CUDA kernels. If it is known that all individual kernels are functioning as expected, it follows that the code as a whole should function as expected as well.

Once the core of the WARP code has been restructured and unit tests implemented such that WARP is more robust and correct, delta-tracking may be revisited. The implementation discussed here should be tested against the fixed code to verify correctness and measure speed. If this delta-tracking method is still slower than ray tracing for the majority of configurations, an alternative algorithm may be developed. The OptiX framework may be leveraged to provide the information needed for the delta-tracking method using only one trace, which could allow delta-tracking to perform better than ray tracing.

Appendix A

New "Where Am I" Algorithm

This appendix includes a disscussion of the new particle location and material update ("where am I") algorithm in the WARP code. The discussion is included here as an appendix because this algorithm was updated after the original presentation corresponding to the master's degree and this report.

The original algorithm, discussed in Section 2.2, was found to produce incorrect results when testing the experimental material and geometry configurations discussed in Section 4.3. Because of this, the particle location update algorithm was modified to be able to correctly handle such configurations in addition to the ones originally tested.

WARP now uses "cell sense" in its ray tracing implementation to determine in which cell and material a neutron is located. Cell sense is positive if a neutron is outside of a cell and negative if a neutron is inside of a cell; it is the product of the surface senses of the cell's constituent planes. Figure A.1 shows an illustration of the process.

In WARP, the surface normal vectors always point outward and the signs of the normals encountered along a trace are summed in the tracing process. When the sum becomes negative, the trace is stopped, and the last intersected cell (and its corresponding material) is the one in which the neutron is located.

Previously, a list of cell numbers was stored for each cell surface intersection that occurred along the trace with the double entries removed to make a nested list of the cell(s) in which a neutron was located. In the new algorithm, cell sense is calculated on the fly as the query ray traverses the configuration geometry, allowing the trace to stop as soon as the cumulative sense becomes negative. This is possible because WARP requires that all geometry surfaces be closed. Additionally, compared to the prior scheme of storing and operating on a list of integers, this new method only requires a single integer to be stored and operated on.

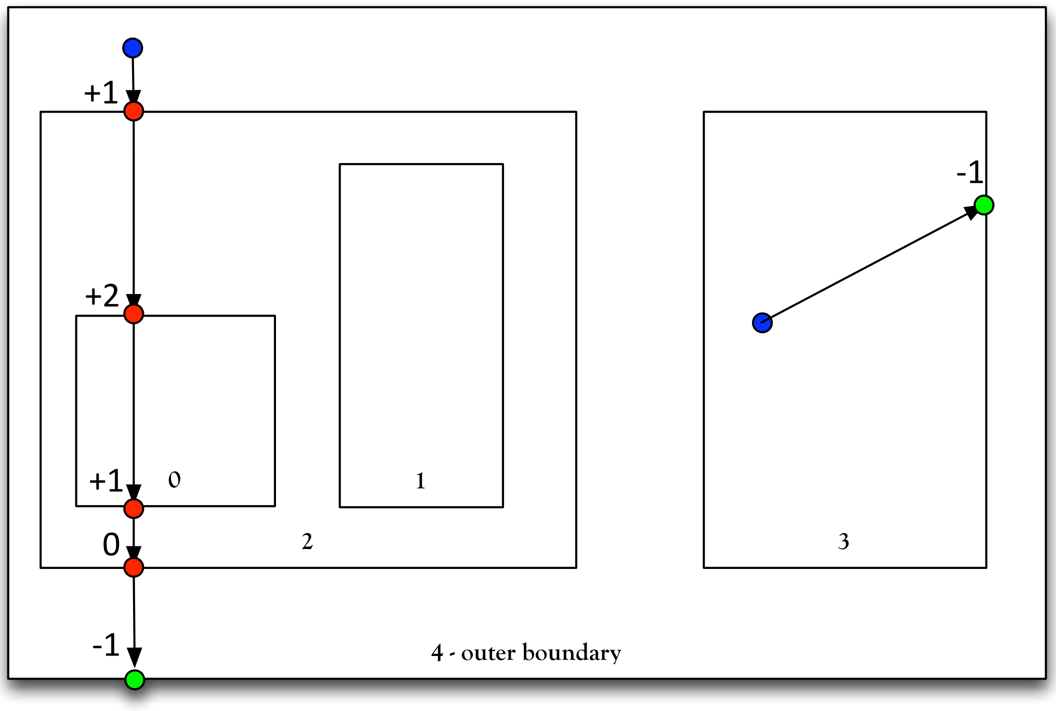

Figure A.1: The new, improved point-in-polygon "where am I" algorithm.

Bibliography

[1] Ryan M. Bergmann and Jasmina L. Vujić. “Algorithmic choices in WARP–A framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs”. In: *Annals of Nuclear Energy* 77 (2015), pp. 176–193.

[2] Oak Ridge Leadership Computing Facility. *CAAR: Call for Proposals*. 2014. URL: `https://www.olcf.ornl.gov/summit/caar-call-for-proposals/`.

[3] Lawence Livermore National Laboratory. *Sierra Advanced Technology System*. URL: `http://computation.llnl.gov/computers/sierra-advanced-technology-system`.

[4] TOP500.org. *November 2014*. 2014. URL: `http://www.top500.org/lists/2014/11/`.

[5] X-5 Monte Carlo Team. *MCNP - A General Monte Carlo N-Particle Transport Code, Version 5*. Volume I: Overview and Theory. (Revised 2/1/2008). Los Alamos National Laboratory, Los Alamos, NM, Apr. 2003.

[6] J Leppänen. *Serpent User Manual*. Tech. rep. retrieved 15/03, 2014.

[7] Los Alamos National Laboratory. *MCNP - Frequently Asked Qustions*. 2014. URL: `https://laws.lanl.gov/vhosts/mcnp.lanl.gov/mcnp_faq.shtml`.

[8] Jaakko Leppänen. “Performance of Woodcock delta-tracking in lattice physics applications using the Serpent Monte Carlo reactor physics burnup calculation code”. In: *Annals of Nuclear Energy* 37 (2010), pp. 715–722. ISSN: 03064549. DOI: `10.1016/j.anucene.2010.01.011`.

[9] *OptiX Programming Guide*. v3.0. NVIDIA Co. Santa Clara, CA, Nov. 2012.

[10] E Woodcock et al. “Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry”. In: *Proc. Conf. Applications of Computing Methods to Reactor Problems*. Vol. 557. 1965.

[11] L.A. Ondis II, L.J. Tyburski, and B.S. Moskowitz. *RCP01 - A Monte Carlo Program for Solving Neutron and Photon Transport Problems in Three Dimensional Geometry with Detailed Energy Description and Depletion Capability*. Tech. rep. 2000.

[12] T.M. Sutton et al. *The Physical Models and Statistical Procedures Used in the RACER Monte Carlo Code*. Tech. rep. July 1999.

[13] N.I. Alekseev et al. *The MCU-PD Code*. 2009. URL: `http://mcuproject.ru/eapd.html`.

[14] Yuuichi Morimoto et al. "Neutronic Analysis Code for Fuel Assembly Using a Vectorized Monte Carlo Method". In: *Nuclear Science and Engineering* 358 (1989), pp. 351–358.

[15] Simon D Richards et al. "MONK and MCBEND: Current Status and Recent Developments". In: *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)* (2013).

[16] B. Cochet et al. "Capabilities overview of the MORET 5 Monte Carlo code". In: *Annals of Nuclear Energy* (2014). ISSN: 0306-4549. DOI: `http://dx.doi.org/10.1016/j.anucene.2014.08.022`. URL: `http://www.sciencedirect.com/science/article/pii/S0306454914004071`.

[17] Jaakko Leppänen et al. "Development of a new Monte Carlo reactor physics code". PhD thesis. VTT Technical Research Centre of Finland, 2007.

[18] Ryan Bergmann. "The Development of WARP - A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs". PhD thesis. University of California, Berkeley, May 2014.

[19] NEA Nuclear Science Committee et al. *International Handbook of Evaluated Criticality Safety Benchmark Experiments*. Nuclear Energy Agency, Organization for Economic Co-operation and Development, 1995.