MONTE CARLO SIMULATIONS OF A CANDU 6 LATTICE CELL USING THE GEANT4 TOOLKIT

L.Russell[1], A. Buijs[1], W. Ford[1] and G. Jonkmans[2]
[1] McMaster University, Hamilton, Ontario, Canada
[2] Atomic Energy of Canada Limited, Chalk River, Ontario, Canada

Abstract

In our prior work, we demonstrated that the Geant4 Monte Carlo toolkit can be used to simulate a quasi-stable, time-dependent neutron population by renormalizing the population at regular temporal intervals. However, this method was only demonstrated with simple uranium spheres of varying diameters. The CANDU 6 lattice cell was chosen as an appropriate benchmark for our stabilization method because of its prevalence in current nuclear research. To simulate the lattice cell, periodic boundary conditions were added to the stabilization method so that the calculations could be performed on an infinite lattice. The results of these calculations were evaluated against several established codes, including DRAGON.

1. Introduction

Reactor physics simulations can generally be divided into (largely) deterministic calculations and stochastic calculations. Up to this point, deterministic calculations have been the workhorses of the professional nuclear community because of their relatively quick and cheap calculations. In recent years, continued development in computer processing power has allowed stochastic calculations to become a viable option for nuclear reactor simulations.

Monte Carlo simulations form an important subset of stochastic calculations. In a Monte Carlo particle physics simulation, individual particles are tracked as they move through different materials undergoing nuclear and electromagnetic interactions such as scattering, absorption, and fission. The average behaviour of many particles over time provides comparable results to analogous deterministic calculations. However, the inherent spatial-dependence of Monte Carlo simulations also allows these simulations to track time of flight, and therefore, model transient behaviour with few assumptions.

Before modeling transient behaviour, additional simulation methods needed to be developed, implemented and validated in an appropriate Monte Carlo code. Geant4 (Geometry ANd Tracking 4), a Monte Carlo toolkit provided by the Geant4 Collaboration, is suitable for this task since it was designed to be flexible, modular, open and transparent [1]. While Geant4 is more user-intensive by design than most other Monte Carlo codes because it requires C++ code development for every simulation, it is extensible and adaptable to many different applications [2]. In a prior work, the Geant4 toolkit was used to develop a method of neutron population stabilization so that a time-dependent neutron population could be tracked in any environment, even ones that were highly sub- or supercritical (i.e. environments leading to rapid exponential population loss or gain, respectively) [3]. However, this method was only partially validated for a simple uranium sphere, and thus, the CANDU 6 lattice cell was chosen as a more practical benchmark [4]. Periodic boundary conditions

were developed in Geant4 and added to the stabilization method to allow the simulation of an infinite lattice of CANDU 6 cells.

Section 2 of this paper will cover the basic fundamentals of Geant4 that are necessary to understand the following discussion. Section 3 will briefly describe the neutron population stabilization method, and Section 4 will describe the implementation of periodic boundary conditions alongside the stabilization method so that an infinite lattice of basic reactor cells may be simulated. Finally, Sections 5 and 6 will discuss the implementation of a CANDU 6 lattice cell, and the validation of this benchmark against established nuclear physics codes. In particular, DRAGON was used as a benchmark for both the criticality and neutron spatial distribution of the CANDU 6 lattice cell.

2. Basic Geant4 Simulations

The Geant4 source code consists of libraries of classes and functions written in C++, and is accompanied by nuclear data formatted for use with the Geant4 functions. The user is responsible for adding key components, such as the initial source generator, the simulation geometry and the main driver file in the form of C++ code, which are then compiled with the source libraries to create executable programs [1]. This design decision adds flexibility to Geant4 but it also requires more effort from the user than most nuclear simulation codes.

While the user is required to generate the physical models and the steering code, the basic tracking algorithm is common to all Geant4 simulations (although this may be changed by modifying the source libraries). A simulation begins with N primary particles (*primaries*) and ends when all the particles have been lost either through absorption or by exiting the simulation world. This includes the primaries and any secondary particles that were created during the simulation by the primaries or their descendants. For efficiency purposes, or other concerns, the primaries may be divided into smaller groups of n particles. In general, the history of a group of n particles and their descendants, from creation to loss, is referred to as an *event*, whereas the combined history of all events is referred to as the *run* [1].

Similar to most Monte Carlo codes, tracking in Geant4 is divided into a series of discrete steps, each of which ends in an instantaneous interaction. These interactions are controlled by physics *processes*, and include hadronic interactions such as elastic and inelastic collisions, radiative capture and fission. Crossing a geometric boundary is also an interaction and is controlled by the transportation process [1]. Additionally, the user may also define new processes and add them to the simulation.

To choose which process occurs at the end of a given step, the process manager requests a proposed step length from each process and picks the process that proposes the smallest step length. For the transportation process, the step length is always the distance to the next geometric boundary in the direction of travel of the neutron. For the hadronic processes, the proposed step length is

$$d_{step}^{i} = \frac{\eta_{\lambda}^{i}}{\Sigma_{i}} \tag{1}$$

where Σ_i is the macroscopic cross section of process i and η_{λ}^{i} is the number of interaction lengths left for that process; in other words, the number of steps of length Σ_i^{-1} [5]. When a primary neutron is created, or when a *hadronic* interaction occurs, the value of η_{λ}^{i} is reset using an exponential deviate

$$\eta_\lambda^i = -\log(r) \tag{2}$$

where r is a uniformly distributed random number between 0 and 1 [1]. However, if the step does not end in a physical interaction (e.g. transportation process), then η_λ^i is decremented using [5]

$$\eta_\lambda^i = \eta_\lambda^i - \frac{actual\ step\ length}{\Sigma_i^{-1}} \tag{3}$$

This preserves the continuity of the particle's history by keeping processes with arbitrary step limits from affecting the physics of the simulation.

The tracking algorithms used in Geant4 differ from other Monte Carlo codes such as MCNP because of the various step limiting processes. In particular, Geant4 allows charged particles to be tracked in a magnetic field, where the path of the particle is made up of short chords approximating the actual curved path of the particle due to the electromagnetic interactions [1]. If Geant4 did not use the tracking algorithm described previously, then the approximation of a curve as a series of chords would again affect the outcome of the simulation. Codes that do not model electromagnetic interactions and do not have step limiters, such as MCNP, can simply calculate the step size as

$$d_{step} = -\Sigma_t^{-1}\log(r) \tag{4}$$

where Σ_t is the total macroscopic cross section of the particle and r is a uniformly distributed random number between 0 and 1 [2].

3. Stabilized Real-Time Simulations

In prior work, it has been shown that Geant4 can simulate time-dependent neutron populations, even in highly super- or subcritical mediums, by renormalizing the neutron population at regular intervals. The major components of this stabilization and how they interact with the basic Geant4 simulation will be described in this section. A more complete discussion of the stabilization process may be found in the referenced paper; this section has been included for the sake of completeness and clarity [3].

The first step is to divide the simulation into discrete time intervals. In Geant4 parlance, the simulation becomes a series of runs that occur sequentially in time. To stop the neutrons precisely at the end of a run, an additional process was created that limited step size to the maximum distance the neutron could travel until the end of the run. That is

$$d_{max} = \left((t_0 + T) - t_{i-1}\right) v(t_{i-1}) \tag{5}$$

where t_0 is the time at the beginning of the run, T is the (temporal) length of the run, t_{i-1} is the current time at the start of the step i, $t_0 + T$ is the time at the end of the run, and $v(t_{i-1})$ is the velocity of the neutron at t_{i-1} [3]. Therefore, this step limiting process always occurs when a neutron reaches the time limit of the run. The neutron is then killed so that the run can end.

For a single run, four important quantities are calculated. First is the run multiplication constant, k_{run},

$$k_{run} = \frac{N(t_0 + T)}{N(t_0)} \tag{6}$$

which simply calculates the absolute population change over the course of the run. Second is the average neutron lifetime, which is calculated by averaging the lifetimes of all the neutrons that were killed in the current run. The neutron lifetime, $\bar{l}$, is defined as the time from the birth of the neutron until it is killed either by escaping the simulation geometry or by absorption. Third is the true multiplication constant, k_{eff}, which is defined as [6]

$$k_{eff} = \frac{rate\ of\ production \times T}{rate\ of\ loss \times T} = \frac{total\ produced}{total\ lost} \tag{7}$$

Note that the two multiplication constants will only be equal if the run duration is equal to the average neutron lifetime. Finally, the Shannon entropy is calculated to determine whether the spatial source distribution of neutrons has converged. The Shannon entropy is defined as

$$S = -\sum_i P(i)\ \log_2\big(P(i)\big) \tag{8}$$

where i denotes an element of a three-dimensional mesh spanning the simulation geometry, and $P(i)$ is the probability of a fission occurring in element i during the run [2]. In practice, the Shannon entropy is calculated by recording the location of each fission during the run and then applying Equation 9. If the spatial source distribution converges, so will the Shannon entropy.

The neutrons that reach the end of a run, and are killed by the step limiting process, are the *survivors* of the run. To transition from one run to the next, these survivors become the primaries in the next run. Therefore, before these neutrons are killed in the current run, their transport parameters must be recorded so that they can be recreated in the next run. This includes the lifetime, position and momentum of the neutron, as well as the η_λ^i values for the four hadronic processes. As mentioned in Section 2, processes that arbitrarily limit the step size should not affect the physics of the simulation, so it is crucial that the η_λ^i values are preserved and are not reset when the neutron is recreated in the next run [3].

The neutron population also needs to be stabilized through renormalization. This is done by either duplicating or deleting survivors to reach the original number of primary neutrons. Whether neutrons are duplicated or deleted depends on whether the medium is sub- or supercritical respectively. The deletions and duplications occur randomly across all the survivors uniformly so that the renormalized population is not biased. Although the neutron population is renormalized at the beginning of each run, the total population change at the end of any run can by calculated by

$$N(mT) = N_0 \prod_{i=1}^{m} k_{run}(iT) \tag{9}$$

where m is a positive integer and mT is the time at the end of the run of interest and N_0 is the initial number of primaries [3].

4. Simulating an Infinite Lattice

A relatively simple, but practical, reactor physics model is the lattice cell of a nuclear reactor. Simulating an entire reactor requires a very complicated geometrical description and significant computational resources for any computation; whereas, simulating a single lattice cell in an infinite lattice is simple by comparison. However, to model an infinite lattice in a Monte Carlo simulation, the lattice cell must have periodic boundary conditions at all six bounding surfaces of the lattice cell. In other words, particles that leave the cell must reappear immediately on the opposite side of the cell.

Periodic boundary conditions were implemented in Geant4 by creating a new process, the boundary step limiter process. This process proposes a step size of zero for any neutron leaving the lattice cell, and a step size of $\sim 10^{302}$mm in all other cases. To determine whether a neutron has left the lattice cell, the boundary step limiter process acts on neutrons when they take their first step outside the cell boundary. In addition, the neutron is leaving the lattice cell if

$$\hat{n}(x,y,z) \cdot \hat{p}(x,y,z) > 0 \tag{10}$$

where (x,y,z) is a point on the surface of the lattice cell and the current position of the neutron, $\hat{n}$ is the outward normal of the lattice cell, and $\hat{p}$ is the momentum direction of the neutron.

When the boundary step limiter acts on a neutron, it creates an identical neutron, as a secondary, on the opposite side of the lattice cell, and then kills the original neutron. Only the position changes, so the secondary neutron is now entering the lattice cell. A secondary neutron is created rather than "teleporting" the original neutron so that the original neutron does not undergo an arbitrarily large displacement *after* the step. As with the step limiter process that kills neutrons at the end of the run, the boundary step limiter process also needs to preserve the η_λ^i values for the four hadronic processes.

5. CANDU 6 Lattice Cell Specification

The CANDU 6 lattice cell is a common benchmark used in Canada. This test case was selected to show the applicability of this Geant4 simulation method. Unlike the original test geometry (U235 spheres), the CANDU 6 lattice cell includes multiple materials at different temperatures. Additionally, the lattice cell emphasizes thermal neutrons as opposed to the high percentage of fast neutrons found in the pure U235 spheres.

The material composition and geometric specification for the lattice cell used in this work was copied from a DRAGON input file where the fuel is fresh, natural uranium oxide (UO_2) [4]. Since Geant4 simulates particle interactions in three-dimensions, whereas the comparison code (DRAGON) used a two dimensional lattice cell, the lattice cell used in Geant4 was uniform in the z-direction. Therefore, the model ignored features such as the fuel rod end caps, the bundle end plates and the spacing between bundles. Other smaller features, such as the fuel element spacers and the graphite lubrication used in the fuel elements (CANLUB), were also ignored for simplicity and so that the Geant4 lattice cell matched the two-dimensional DRAGON model.

Figure 1 shows the lattice cell used in this paper. The standard lattice pitch of 28.575 cm was used in the x- and y- axis, and the cell length in the z-axis is equal to the length of a standard CANDU 6 bundle, 49.53 cm [4]. Note that the cell length is arbitrary and could have been chosen to be any reasonable length since the lattice cell is uniform in the z-direction.

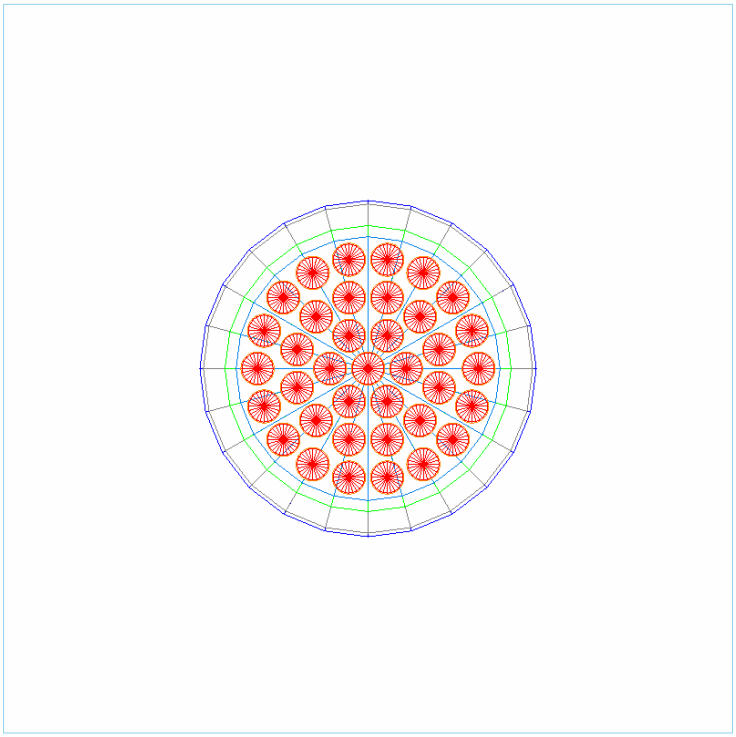

Figure 1 The standard CANDU6 lattice cell.

In Figure 1, the different materials and volumes are colour coded along the outer edge of each volume. The lattice cell is composed of a fuel channel surrounded by heavy water moderator (light blue in the figure above). The fuel channel consists of two concentric tubes, the calandria tube (dark blue) and the pressure tube (green), which are separated by an annulus filled with carbon dioxide (grey). The pressure tube contains the fresh, natural uranium oxide fuel (red) and the heavy water coolant (light blue). The fuel itself is sheathed in metal that is composed mostly of zirconium (orange). Note that the radial spokes seen on the cylindrical volumes are simply artefacts of the rendering process.

6. Results

The simulations that provided the results below used 100,000 primary neutrons starting from the centre of the central pin with energies sampled from a Gaussian distribution centred at 1 MeV. The simulations lasted for 300 runs of 100 μs and the first 100 runs were disregarded when calculating the average k_∞ values. For simplicity, the delayed neutrons were born at the time of fission, but their lifetime was set to the time they would have been born after the fission. That is

$$l_d = t_{birth} - t_{fission} \tag{11}$$

where l_d is the delayed neutron lifetime, $t_{fission}$ is the time of the fission that spawned the delayed neutron precursors which eventually produced the delayed neutron, and t_{birth} is the (future) time at which the delayed neutron should have been born. Thus, the delayed neutrons were produced instantaneously but their characteristics were sampled from the delayed neutrons distributions for energy, momentum and lifetime.

6.1 Comparison of lattice cell criticality

The criticality of the standard CANDU 6 lattice cell was calculated using the simulations described above. These results were compared to published results from deterministic codes, as well as a criticality calculation in DRAGON. Table 1 below compares the results for k_∞ from the relevant simulations.

Table 1 Infinite lattice criticality estimates from various nuclear code simulations

Simulation Code	k_∞	Notes
GEANT4	1.128	3D Monte Carlo code with continuous energy
DRAGON	1.12418	2D Deterministic code with 69 energy groups
SCALE - NEWT	1.12856	2D Deterministic code with 238 energy groups [7]
SCALE – KENO VI	1.13030	3D Monte Carlo code with continuous energy [7]

The estimate for k_∞ calculated using Geant4 is between the values calculated by the other codes shown in Table 1, and thus, the criticality calculation in Geant4 is comparable to other more established codes for the standard CANDU 6 lattice cell. Some discrepancies should be expected since the nuclear data varies for each code. In addition, only the Geant4 and KENO results were calculated using continuous energy data libraries; the results calculated with DRAGON and NEWT used 69 and 238 energy groups respectively [4,7].

6.1.1 Two dimensional flux distribution

In addition to the k_∞ estimate from each code, the neutron density along the horizontal centreline of the lattice cell was compared for the Geant4 and DRAGON simulations. In practice, the centreline is represented by a 2 cm thick rectangular region that is bounded by x-z planes at y = ±1 cm. For the three-dimensional cell used in Geant4, the z-direction is ignored; that is, any discretization in x and y extends the full length of the cell in z. The neutron density from Geant4 represents a snapshot in time; that is, the density of simulated neutrons in the lattice cell at a time t. Similarly, DRAGON calculates the average flux values for each defined region in the lattice cell, where each region represents a discrete homogeneous section of the lattice cell created by the discretization of the lattice cell in DRAGON.

To compare the flux from DRAGON to the instantaneous density from Geant4, the flux in DRAGON was calculated for each region and each energy group. Therefore, for each region, 69 average flux values were calculated. These values were converted to neutron densities using

$$n_g^i = \frac{\phi_g^i}{v_g} \tag{12}$$

where n_g^i is the density of energy group g neutrons in region i in square centimetres (two-dimensional), ϕ_g^i is the corresponding flux value, and v_g is the average neutron velocity for the energy group [6]. This average velocity was derived using the following expression

$$v_g = \sqrt{\frac{(E_g^+ + E_g^-)}{m_n}} \tag{13}$$

where E_g^+ and E_g^- are the upper and lower limits of the energy range spanned by group g, and m_n is the mass of a neutron. Thus, the neutron density of each region was calculated by summing the individual contributions from each energy group $\left(n^i = \sum_g n_g^i\right)$. Since the magnitude of the neutron density is arbitrary, the DRAGON density values were all scaled by a constant factor to match the Geant4 data; the constant multiplicative factor was determined using a least-squares approximation.

Figure 2 shows the comparison between the neutron densities in DRAGON and Geant4. The statistical error in Geant4 is shown through the error bars, while the error in DRAGON was small, and thus, was neglected in the figure (the maximum error in the flux values was less than 0.5%). Additionally, the relevant structures along the centreline of the lattice cell are delineated by the hatched areas in Figure 2. While the third ring of fuel elements is offset by 0.262 radians and does not intersect the horizontal centreline, it does influence the neutron flux in the surrounding coolant. Therefore, this third ring of elements is represented in Figure 2 by a sparsely hatched region where the hatching marks are 90° out of phase from the rest.

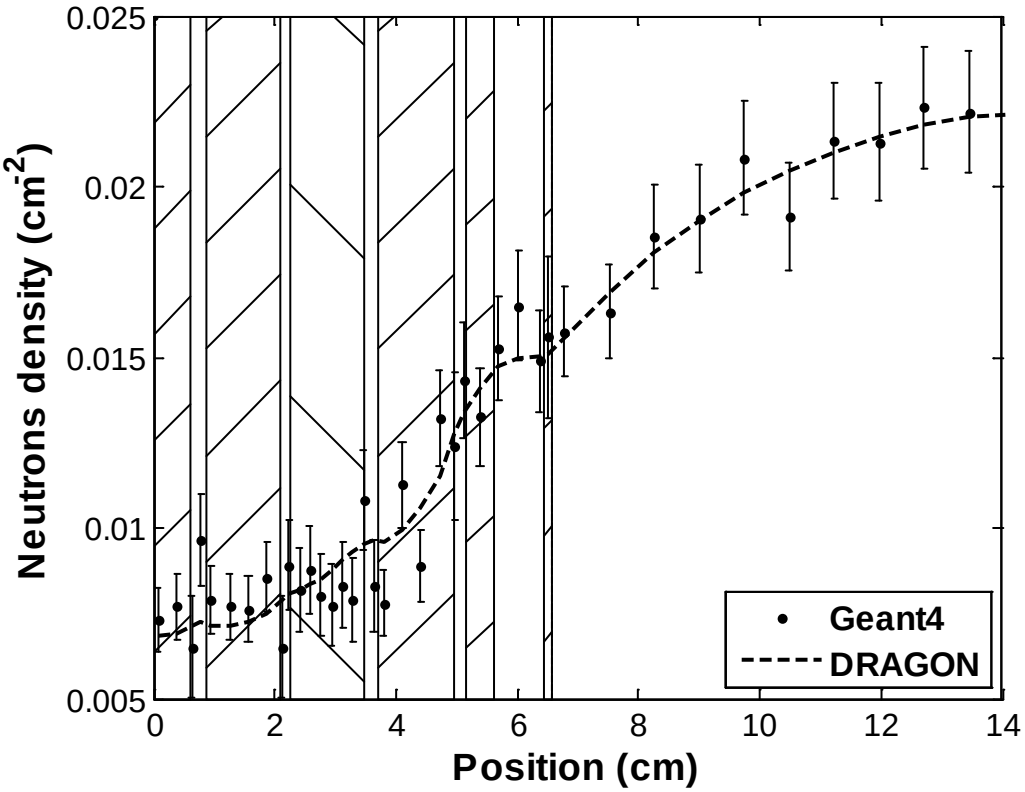

Figure 2 Centreline neutron density of the CANDU 6 lattice cell for Geant4 and DRAGON.

In general, the DRAGON and Geant4 neutron densities match within error. To reduce the stochasticity of the Geant4 results, the neutron density values were derived from three different snapshots in time. The snapshots were taken at 25, 30 and 35 ms, which corresponds to 250, 300 and 350 runs respectively. All of these snapshots occurred after the neutron population had converged to a stable spatial distribution so the difference in time is inconsequential. Some discrepancies are expected because the error only accounts for the statistical error in the Geant4 simulation. Discrepancies in the nuclear data and the simulation methods will result in some finite error.

6.2 Criticality of lattice cells with varied lattice pitches

To further validate the applicability of the Geant4 model described above, the criticality of the lattice cell was calculated at five more lattice pitches. These results are shown in Figure 3 along with estimates from DRAGON using the same geometries. Other than having different lattice pitches, these simulations were identical to the simulation used to calculate the criticality of the standard CANDU 6 lattice. The final Geant4 k_∞ values were calculated from simulations of 250 runs, where the first 50 runs were disregarded when calculating the average k_∞ for each lattice pitch.

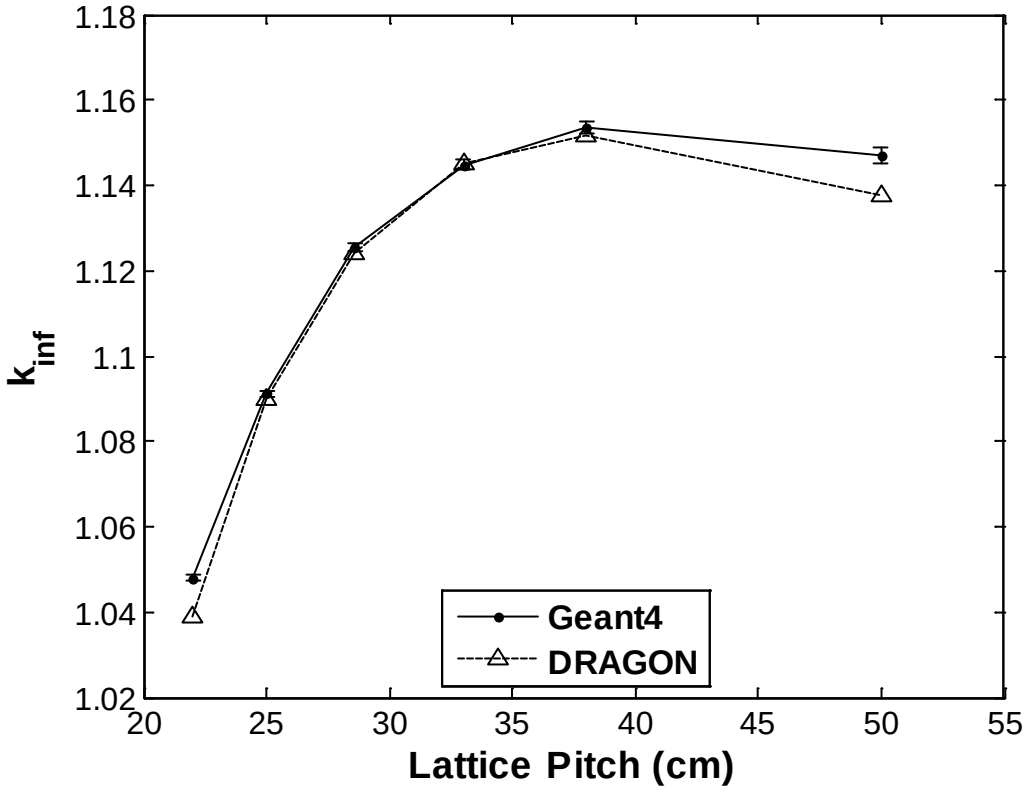

Figure 3 Estimates of for CANDU 6 lattice cells with varying lattice pitch.

Again, the error bars show the statistical error in the criticality estimates, and the statistical error reported by DRAGON was too small to be relevant on the plot. The Geant4 values agree with the DRAGON estimates within 10 mk, which is used as an arbitrary limit to account for all possible errors. Moreover, both codes predict a maximum reactivity at a lattice pitch of approximately 38

cm. The deviation observed at the minimum and maximum pitches could be a result of the differing importance of the moderator in the cell and the particular spatial discretization used in each DRAGON simulation.

7. Conclusion

In our prior work, we have shown that the Geant4 Monte Carlo toolkit can be used to develop a stochastic simulation that tracks neutron populations in time regardless of the medium. Our simulation method renormalizes the neutron population at regular intervals so that the population remains manageable even in strongly sub- or supercritical mediums. Prior to the work presented in this paper, the neutron tracking and stabilization method was only validated for very simple geometries. A more practical and well known benchmark is the CANDU 6 lattice cell.

Before simulating the lattice cell, we developed a physics process for Geant4 to implement periodic boundary conditions. With this process in place, the CANDU lattice cell could be simulated with a time-dependent neutron population and analysed for important characteristics. Foremost among these characteristics were the infinite lattice multiplication constant, k_{∞}, and the centreline neutron flux. The multiplication constant agreed within 10 mk of the other standard nuclear codes, and the centreline neutron density predicted by Geant4 and DRAGON generally agreed within the statistical error of the Geant4 values. Finally, to ensure that our simulation method was robust, a sensitivity analysis was performed on the lattice pitch of the CANDU 6 cell in both Geant4 and DRAGON. At all six lattice pitch values simulated, the Geant4 results agreed with DRAGON within 10 mk.

8. References

[1] S. Agostinelli and et al, "GEANT4: A simulation toolkit," Nucl. Instrum. Meth. A, vol. 506, pp. 250-303, 2003.

[2] X-5 Monte Carlo Team, "MCNP Volume I: Overview and Theory," Los Alamos National Labs, MCNP 5 User Manual, LA-UR-03-1987, 2008.

[3] L. Russell, G. Jonkmans, and A. Buijs, "A Method for Simulating Real Time Neutron Populations using the GEANT4 Monte Carlo Toolkit," in Proceedings of the 33rd Annual Conference of the Canadian Nuclear Society, Saskatoon, Saskatchewan, Canada, 2012.

[4] G. Marleau, A. Hebert, and R. Roy, "A User Guide for DRAGON 3.06," Ecole Polytechnique de Montreal, Technical Report IGE-174 Rev. 10 2012.

[5] GEANT4 Collaboration, The GEANT4 Toolkit, 2011, v.4.9.5.

[6] J. J. Duderstadt and L. J. Hamilton, *Nuclear Reactor Analysis*. Mississauga, Ontario: John Wiley and Sons, Inc., 1976.

[7] M. R. Ball, A. C. Morreale, D. R. Novog, and J. C. Luxat, "The Effect of Super-Cell Calculations due to Modeling CANDU Fuel Pin Clusters as Annuli," in Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Rio de Janeiro, RJ, Brazil, 2011.