# Parallelization of Neutron Transport Approximation via Monte Carlo

**Fawaz Ali, B.Eng (Hons.)**
Graduate Student, Faculty of Science
University of Ontario Institute of Technology, Oshawa, ON, Canada
E Mail: fawaz.ali@mycampus.uoit.ca

**Dr. Ed Waller, P.Eng., CAIH, CHP**
Associate Professor, Faculty of Energy Systems and Nuclear Science
University of Ontario Institute of Technology, Oshawa, ON, Canada
E Mail: Ed.Waller@uoit.ca

## Abstract

This paper discusses the parallelization of a Monte Carlo-based neutron random walk FORTRAN code in $D_2O$ (heavy water). In nuclear environments, proper selection of material is crucial is minimizing the radiation dose that an individual receives upon exposure (this material is placed between the radioactive source and the individual). Thus, the knowledge of how radioactive particles move through such materials is of great importance since one can estimate how many of such particles can penetrate a material. Neutrons will be the radioactive particle that will be analyzed since they represent one of the most important types of radiation in nuclear environments. The motivation to parallelize a Monte Carlo-based neutron random walk code lies in the fact that actual neutron sources vary in strength, ranging from the order $10^2$ to $10^{14}$ neutrons. As such, when simulating this many particles in a serial code, the computation time will be extensive. Thus, the focus of this paper is to document the parallelization of the abovementioned code.

## 1.0 Introduction

In the field of Nuclear Engineering, the issue and concept of Nuclear Radiation is of great importance. This field of engineering has a tremendous amount of applications and as such, the radiation produced by such applications is rigorously studied. A central theme of such studies is the computation of the dose received by an individual exposed to nuclear radiation. These calculations can be carried out by numerous Monte Carlo-based radiation physics codes. A typical investigation that is conducted with the aid of Monte Carlo techniques is shown below:

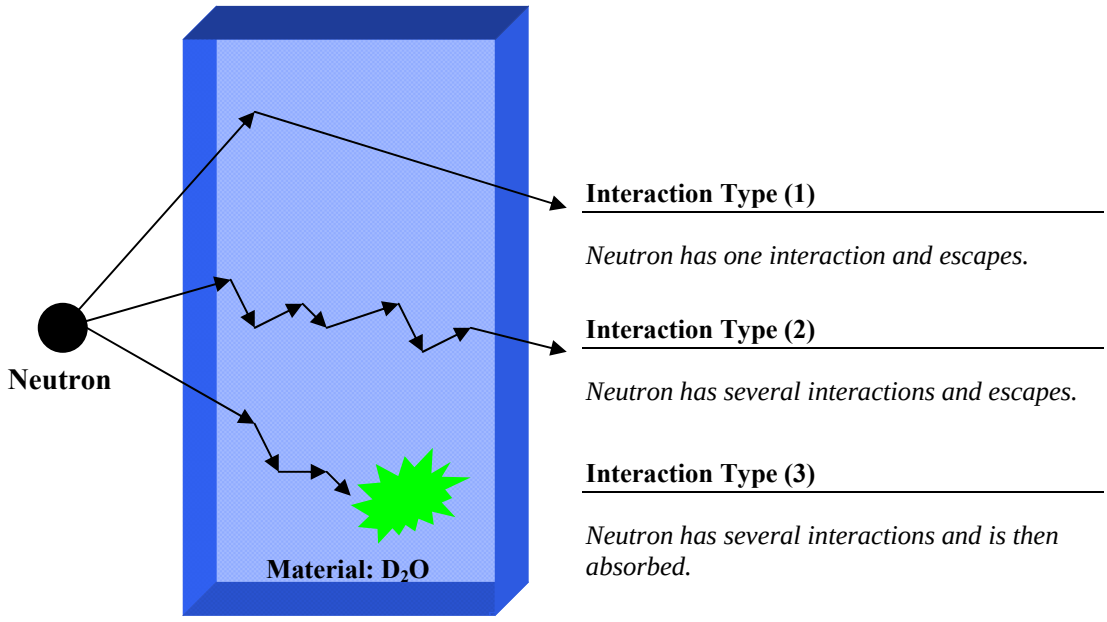


**Figure 1** ***Illustration of a neutron interaction with a two-dimensional $D_2O$ slab [1]***

Figure 1 highlights the motivation to study such a topic. This figure illustrates a neutron source that is to the left of a $D_2O$ slab. In nuclear environments, such materials serve to protect an individual from interacting with a radioactive source because this material is placed between the individual and radioactive source. Thus, the analysis of this project will allow one to analyze how effective a material is at reducing the number/intensity and energy of neutrons that pass through it and interact with the individual - this phenomenon is known as *attenuation*. The aim of the configuration shown in Figure 1 is to therefore attenuate the number of neutrons moving through the material such that the individual standing at the other side of the material will interact with as small amount of neutrons as possible.

## 2.0 Why Parallelization?

As mentioned in the abstract, the simulation of the movement of millions of neutrons in a material using a *serial code* (by serial, it is meant a code that runs on one processor) will result in extensive computation times. This gives rise to the need for parallel computing since the use of multiple processors will result in significantly decreased run times. In fact, in an ideal simulation, if **n** processors are used, the parallel computing time will be smaller than the serial computing time by a factor of **n**. However, this is not always the case since the *communication overhead* (which is essentially the time it takes for different processors to talk to each other to exchange information) will make a small but measureable contribution to the run time. As a result, the parallel computing time will be smaller than the serial computing time by a factor that is slightly smaller than **n**, as per the analogy above.

The process of transforming a serial code to a parallel code is called *parallelization*. When parallelizing a code, one must add provisions to allow for the code to multitask the computation and for the processors to exchange information. Such provisions are made possible by the availability of ***Message Passing Interface (MPI)*** functions. It is important to note that parallelization of a code results in work being distributed amongst processors whereas the MPI functions seek to consolidate the results of the computation performed on each processor.

Lastly, it is of note that the serial and parallel code has been executed using the SHARCNET (Shared Hierarchical Academic Research Computing Network) clusters available to UOIT.

## 3.0 Problem Statement

The purpose of this section is to provide a description of the problem that will be examined. Essentially, the intent is to analyze neutrons emitted from an isotropic neutron source in the x-y plane. This source can emit varying amounts of neutrons ranging from $10^5$ neutrons to $10^7$ neutrons. As will be shown in the latter sections of this report, the amount of neutrons that will be emitted will vary within the abovementioned range.

What is sought after is the detection of how many neutrons cross each point along the +x direction starting from x=+1cm to x=+30cm in +1cm increments. The following diagram depicts the abovementioned scenario that will be analyzed:

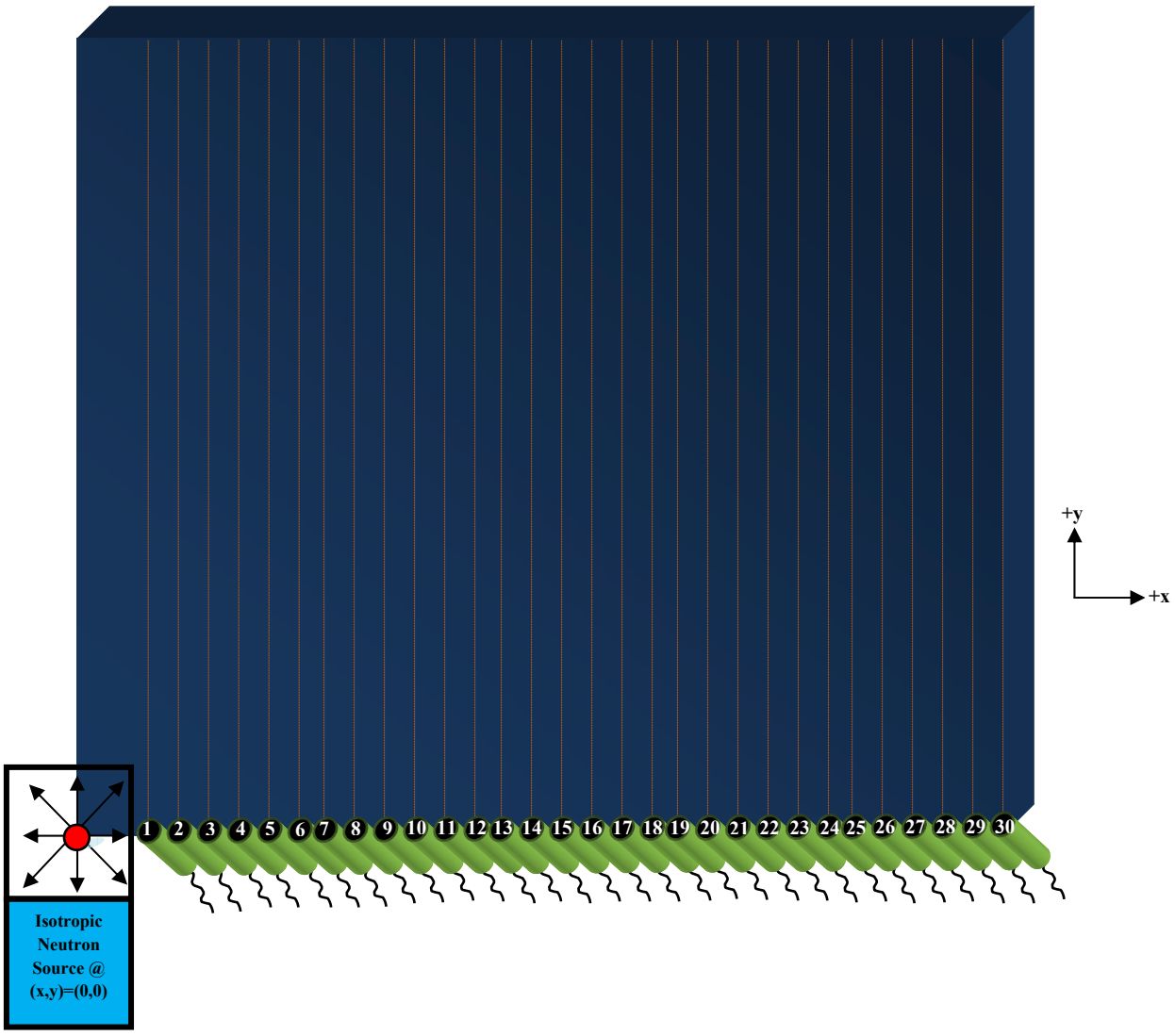


**Figure 2** ***Illustration of configuration to be analyzed***
*The small cylinders placed along the +x axis denote detectors*

Analyzing the above figure will allow for not only the simulation of the movement of neutrons through the material but also the computation of the distribution of the neutron population along the +x axis.

Before any simulation was conducted, several assumptions were made and are summarized below:

- The neutron source is isotropic and is located at the origin of the x-y plane. That is, the initial position of all neutrons is at (x,y)=(0,0)
- The extent of the +x axis is from x=+1cm to x=+30cm in increments of +1cm. Neutron detectors will be placed at each of these positions (thereby positioning 30 detectors along the +x axis) and will only detect forward scatters in the +x direction
- Each neutron undergoes 100 collisions in the material (this will ensure that each neutron will reach thermal energy) and has an initial energy of 2 MeV
- The material that is used will be heavy water ($D_2O$). *This material is used in the CANDU reactors.*
- Elastic scattering will only be considered whereby each neutron will undergo solely scattering reactions. Moreover, each neutron is considered to be "absorbed" (and therefore non-existent) after it attains thermal energy (2.5E-8 MeV) or below
- The following source strengths (i.e. the number of neutrons emitted from the neutron source) will be analyzed

| | |
|---|---|
| $10^5$ | $5x10^5$ |
| $10^6$ | $5x10^6$ |
| $10^7$ | $1.5x10^7$ |

Lastly, the total nuclear cross sections that are employed in this simulation are dependent on the neutron energy, and its distribution is shown in Figure 3.

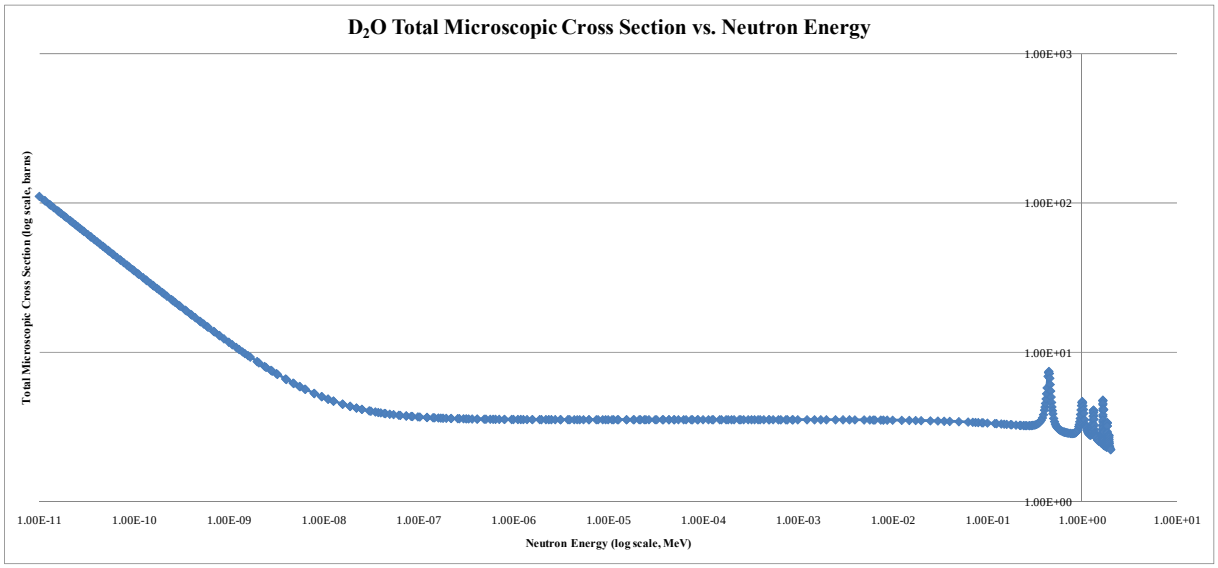


**Figure 3** ***$D_2O$ Total Microscopic Cross Section Spectrum [2]***

### *3.1 Pertinent Parameters*

The central focus of this paper is to compute and analyze the following data:

- **Normalized Neutron Current ($J_+$)**

  This quantity is essentially the number of neutrons passing through a point (in the +x direction) divided by the number of source neutrons. This is stated mathematically:

$$J_+(x) = \frac{Neutron\ Current\ at\ x}{Number\ of\ Source\ Neutrons}$$

- **Fractional Standard Deviation of $J_+$**

  The fractional standard deviation is a metric that quantifies the accuracy of the results. The smaller the values, the more accurate the results are. It is defined as:

$$fsd = \frac{Standard\ Deviation}{Mean\ Value}$$

- **Speedup**

  Speedup is a metric that is used to measure the performance of the parallelized code. Speedup is defined as ([3]):

$$Speed\ Up\ (n) = \frac{Run\ Time\ of\ the\ Serial\ Program}{Run\ Time\ of\ the\ Parallel\ Program\ with\ \boldsymbol{n}\ processors}$$

  As mentioned in section 2.0, an ideal parallelized code will result in a speedup of **n**, where **n** is the number of processors used. This is termed *linear speedup*.

- **Figure of Merit (FOM)**

  Figure of Merit is a metric that judges the quality of the parallelized code. It is defined as follows:

$$FOM = \frac{1}{(fsd^2)(Computation\ Time)}$$

  The use of two or more processors will result in a lower computation time, in comparison to the computation time when using one processor. As such, the Figure of Merit will increase. Therefore, it is desired that the as the number of processors increase, so too will the Figure of Merit.

### *3.2 Objectives of the Simulation*

The purpose of this section is to outline what is to be achieved via this simulation. They are summarized as follows:

- For varying source strengths, find the normalized neutron current at each position along +x axis. This indicates how the neutron population is distributed along the +x axis
- The fractional standard deviation decreases as the source strength increases
- As a result of parallelization, linear speedup is expected
- As a result of parallelization, it is expected that the Figure of Merit will increase as the number of processors used increases

### *3.3 Mathematical Models*

The mathematical models used for this simulation have been developed in [1]. These models describe how attributes of a neutron change after colliding with a nucleus of the material that it travels in. Figure 4 illustrates this phenomenon.

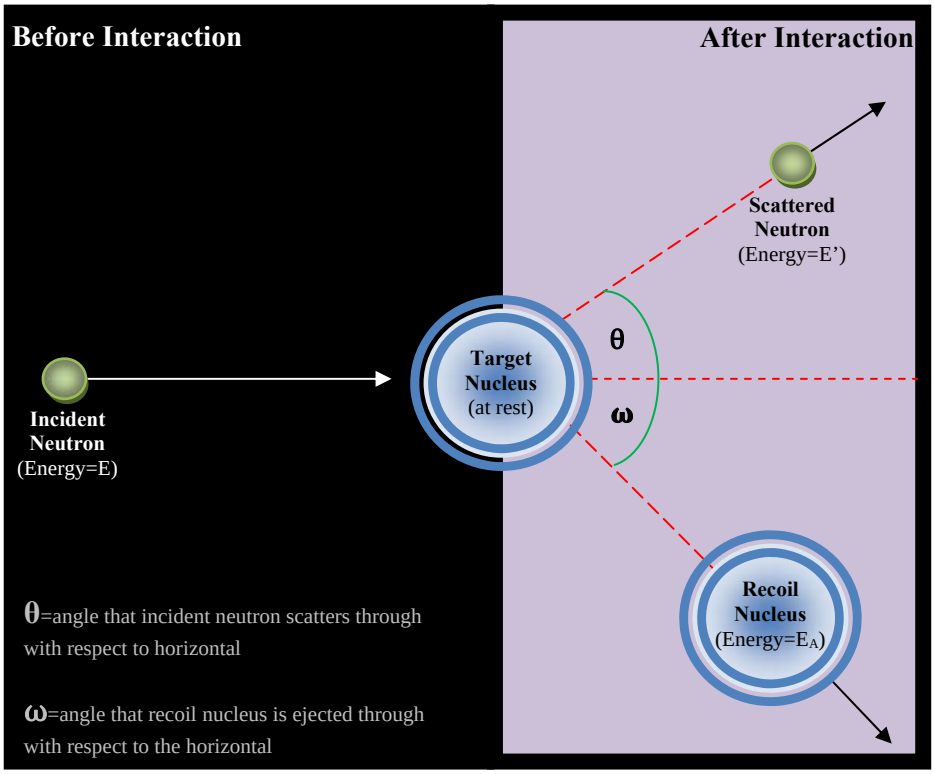


**Figure 4** ***Illustration of interaction of neutrons with material nuclei [1]***

More specifically, these models have been derived by finding relationships between the neutron's position and energy at two collision/scattering points within the material. This is illustrated in Figure 5 below:

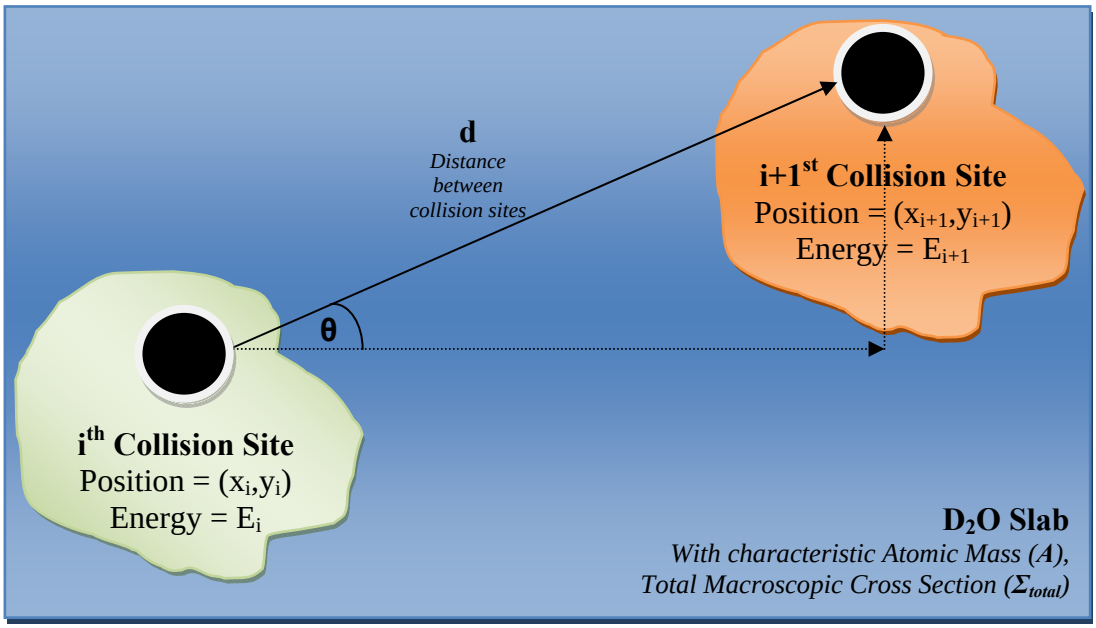


**Figure 5** ***Relationship between neutron properties at different collision sites***

Via Figure 5, the mathematical models used to describe the neutron properties at each collision site are as follows:

$$d = \frac{1}{\Sigma_{total}} \ln(1 - p_1) \quad \textbf{(1)}$$

$$x_{i+1} = x_i + d\cos(\theta) = x_i + d(1 - 2p_2) \quad \textbf{(2)}$$

$$y_{i+1} = y_i + \sqrt{d^2 - (x_{i+1} - x_i)^2} \quad \textbf{(3)}$$

$$E_{i+1} = E_i \left(\frac{1}{(A+1)^2}\right)\left(\cos(\grave{e}) + \sqrt{\cos^2(\grave{e}) + A^2 - 1}\right)^2 =$$

$$E_i \left(\frac{1}{(A+1)^2}\right)\left((1 - 2p_3) + \sqrt{(1 - 2p_3)^2 + A^2 - 1}\right)^2 \quad \textbf{(4)}$$

Note that $\mathbf{p_1}$, $\mathbf{p_2}$, and $\mathbf{p_3}$ represent random numbers that are uniformly distributed on the interval [0,1).

## 4.0 Description of Serial Code

The purpose of this section is to discuss how the serial code works. The description of the serial code is provided in Figure 6.

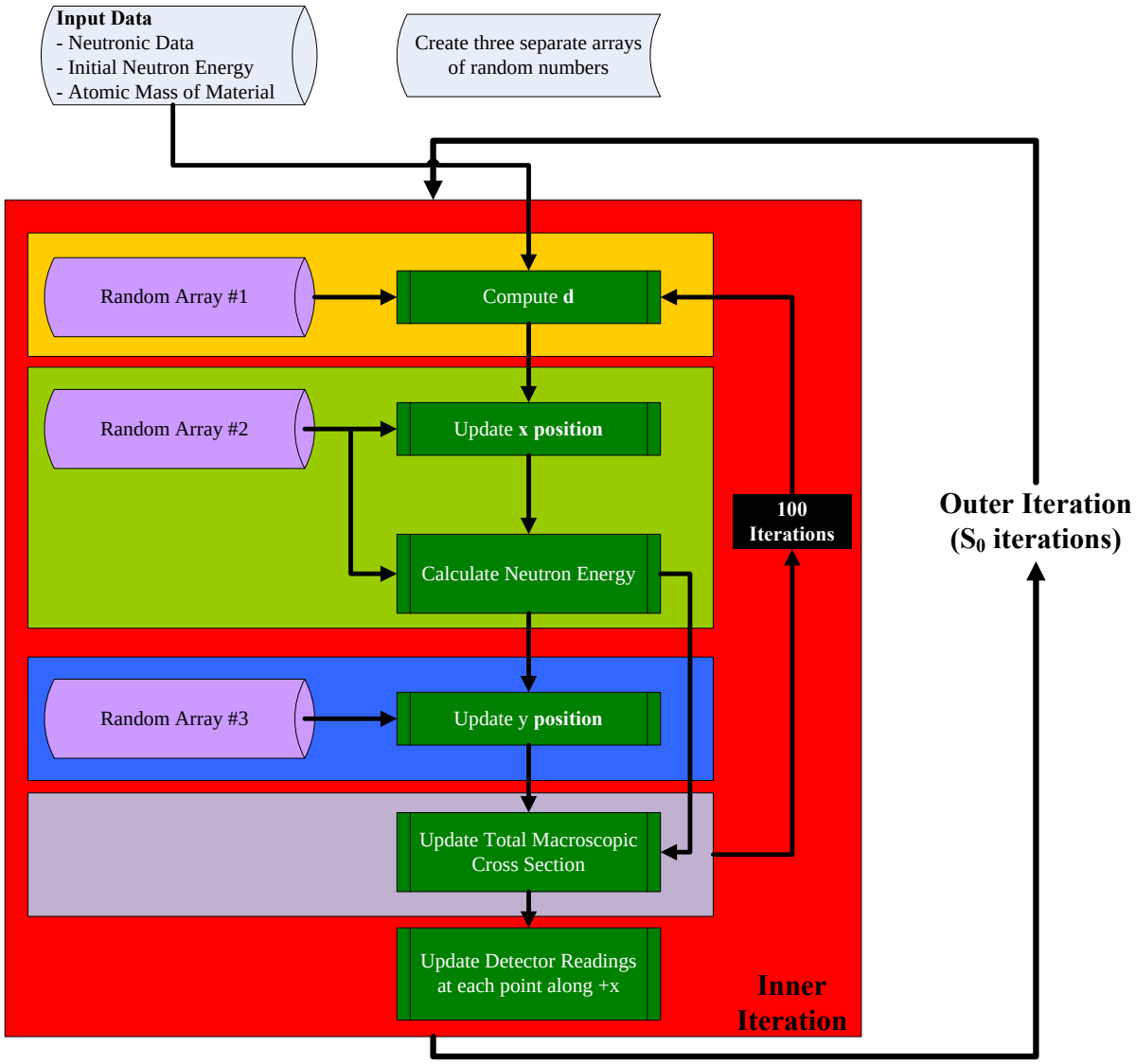


**Figure 6** ***Illustration of serial code***
*$S_0$ stands for the number of source neutrons*

As shown in Figure 6, there are $S_0$ outer iterations. This means that if there are 1 million source neutrons, then there will be 1 million outer iterations. Simply put, the inner iteration tracks each individual neutron while the outer iteration provides the inner iteration a new neutron to track.

## 5.0 Parallelization Methodology

The purpose of this section is to describe how the parallel code was constructed from the serial code.

The foremost task that must be conducted prior to developing the parallel code is determining which part of the serial code is to be parallelized. As described in the abstract, a neutron source can contain a very high amount of neutrons. To simulate the movement of these amount particles will be computationally expensive. Thus, the outer iteration shown in Figure 6 has been chosen to be parallelized.

Essentially, $S_0$ iterations are split up and a varying number of neutrons are sent to each processor to be tracked. This is done via the block distribution method **[3]**. *The purpose of employing this method is to **maximize** the number of processors that can be used to run the code.* The importance of choosing to parallelize the outer iterations lies in the fact that each processor tracks a fraction of the total number of source neutrons. **As such, each processor has its own detector readings at *EACH* point along +x**. This is illustrated in Figure 7.

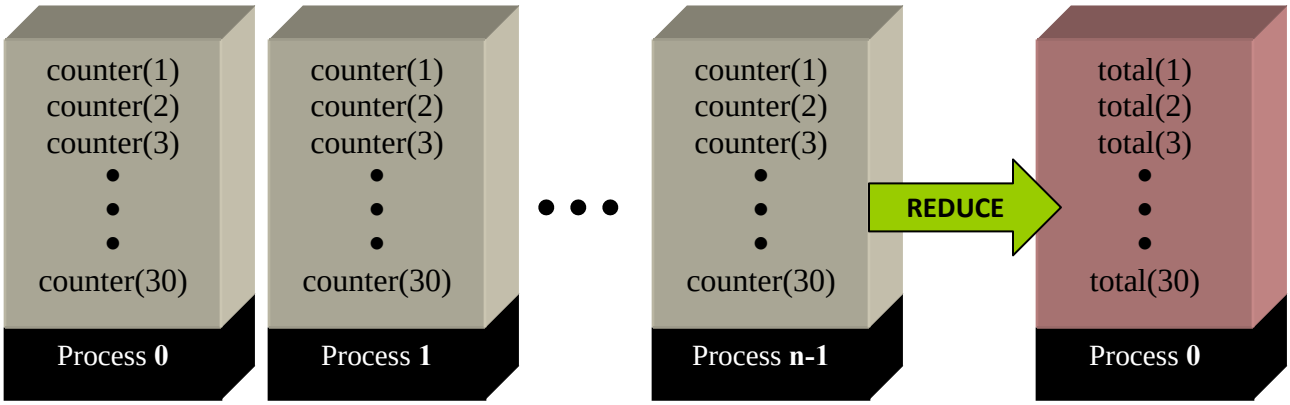


**Figure 7** ***Illustration of parallel code***

As shown in Figure 7, each processor will possess a vector (of length 30, since there are 30 detectors placed along the +x axis) where each entry contains the number of neutrons that have crossed the respective position on the +x axis. This fact therefore allows for the use of the **MPI_REDUCE()** function. The reduce function is used because as illustrated in Figure 7, the intent is to reduce (and sum) all of the detector readings (at each point along the +x axis) from each process to process 0. For example, the intent is to reduce and sum all detector readings at x=+1cm (which is stored in counter(1) entry in each processor) to the first element in the **total** vector, total(1).

Figure 8 shows a screenshot from the FORTRAN code. This illustrates how the MPI_REDUCE() function was implemented.

```
!Step (4) Collect all data from each process to process 0
!-------------------------------------------------------------------------------
!As mentioned above, we will REDUCE each detector reading at each point along +x from each processor
!to the respective entry in total().  This will be done at process 0

  do f =1,max_domain
    CALL MPI_REDUCE(count(f),total(f),1,MPI_REAL,MPI_SUM,0,MPI_COMM_WORLD,ierr)
  end do
```

**Figure 8** **MPI_REDUCE() Implementation**
*Note that MPI_SUM() sums count(f) from each process and stores this sum in total(f)*

### *5.1 Block Distribution Method*

The block distribution method, as outlined in [3], is based on the division of the number of outer iterations and the number of processors. This division results in a ***quotient*** **(q)** and ***remainder*** **(r)**.

This method goes on to state that:

- *Processes **0** to **r-1** will be allotted **q+1** iterations each*
- *The rest are assigned **q** iterations*

In the FORTRAN code, the block distribution is implemented as a subroutine. This method essentially calculates how many iterations each process receives and the mechanism that it uses is outlined above.

## 6.0 Physics Results

The purpose of this section is to document the physics results from the code. As stated in section 3.2, the normalized neutron current along the +x axis and the associated fractional standard deviation are to be found. The following figures show these distributions for varying source strengths.

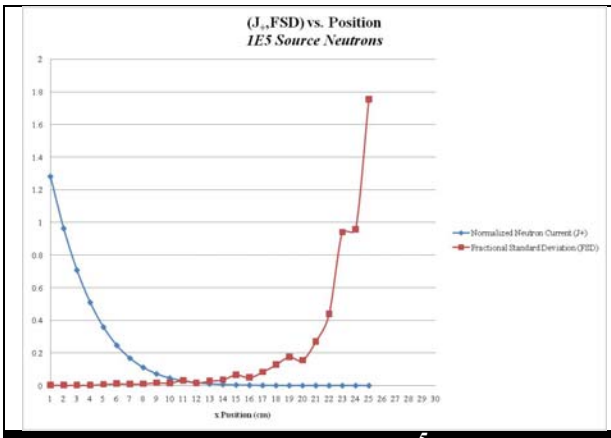


**Figure 9** ***(J+,FSD) for $10^5$ Source Neutrons***

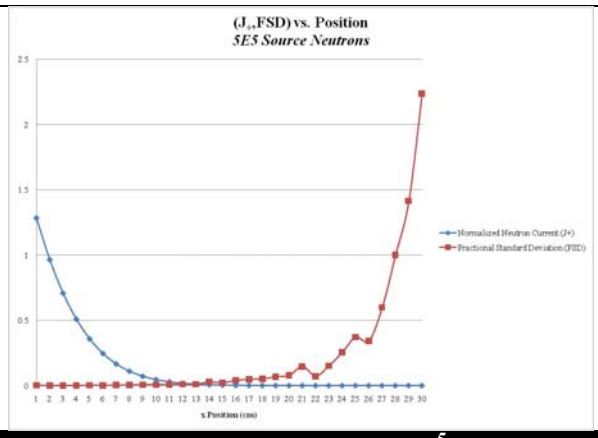


**Figure 10** ***(J+,FSD) for $5x10^5$ Source Neutrons***

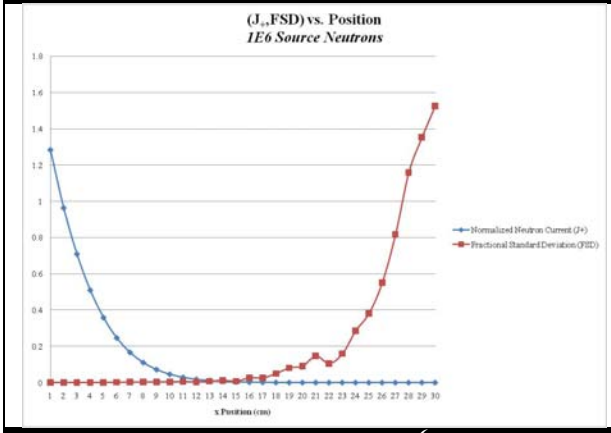


**Figure 11** ***(J+,FSD) for $10^6$ Source Neutrons***

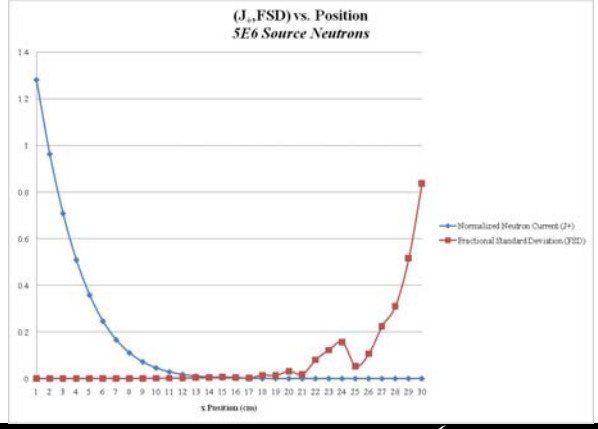


**Figure 12** ***(J+,FSD) for $5x10^6$ Source Neutrons***

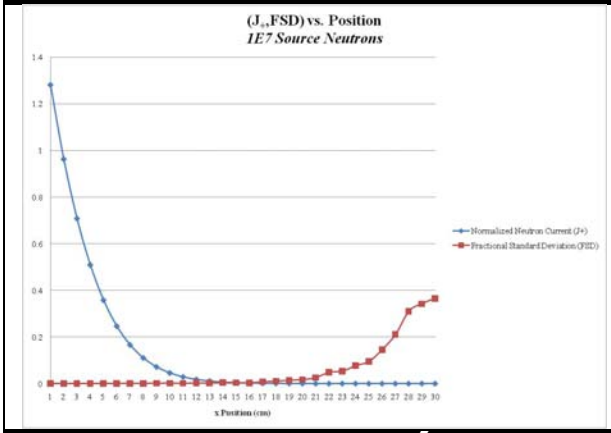


**Figure 13** ***(J+,FSD) for $10^7$ Source Neutrons***

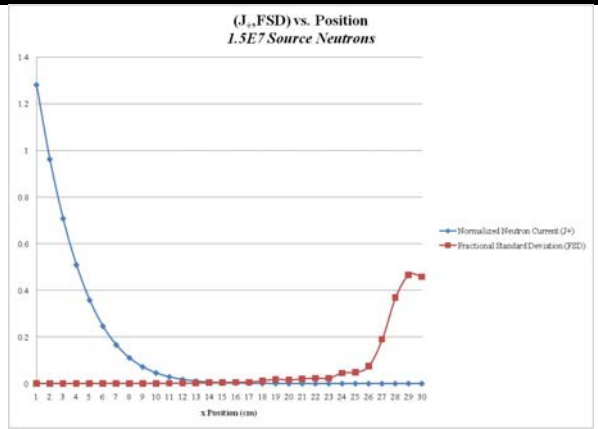


**Figure 14** ***(J+,FSD) for $1.5x10^7$ Source Neutrons***

Figures 9 to 14 display the same trends and in fact, almost the same values for $J_+$ at each point along the +x axis. More specifically, the neutrons appear to be attenuated. In a nuclear engineering context, attenuation means that the number of neutrons that are able to penetrate deeper into a material decreases at an exponential rate (in other words, the plot of the neutron

current as a function of position possess a decaying exponential trend). This is, of course, the case with the abovementioned plots.

The reason why the values of $J_+$ appear to be the "same" in each plot is simply due to the fact that we are dividing (or normalizing) the amount of neutrons that penetrate each point along the +x axis by the number of source neutrons. Thus, if we are to multiply each value of $J_+$ by the number of source neutrons to get the actual amount of neutrons crossing each point along +x, we will see that the values of $J_+$ increase as the source strength increases.

It is also of note that for each of the abovementioned figures, the values of $J_+$ as we move towards the origin (or source) attains a value greater than one. In classical nuclear engineering texts [4], the normalized current should attain a maximum value of one near the source and then decay exponentially as we progress along the +x axis. The reason why in this situation the value of $J_+$ attains a value higher than one in the vicinity of the source is due to the fact that the neutrons are taking a Monte Carlo-based random walk. As a result, the neutrons tend to travel back and forth *about* a particular point on the +x axis. Consequently, the detector at that point will detect a particular neutron more than once. Coupled with this, is the fact that the neutrons are very energetic in the neighbourhood of x=+1cm, since they have been emitted from the *nearby* source, and are likely to undergo both multiple backward and forward scattering reactions without reaching thermal energy. Since we are simulating millions of neutrons, this occurrence will multiply thereby yielding values of $J_+$ greater than one as we approach the source. This phenomenon is depicted in Figure 15.

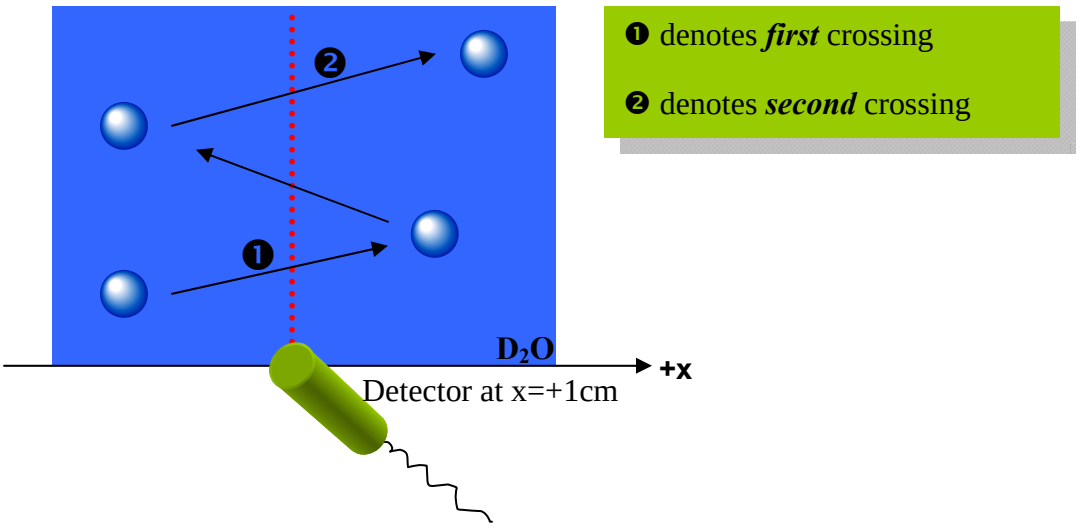


**Figure 15** ***Back-scattering and Forward-scattering of neutrons***

The other results that are of interest are (1) the change in fractional standard deviation as we progress along the +x axis and (2) the change in fractional standard deviation as we increase the number of source neutrons. With respect to the first point, Figures 9-14 illustrate the fact that the fractional standard deviation increases as we move further from the source. This is simply due to the fact that as we move deeper into the material, a smaller amount of neutrons will be able to penetrate it. As such, the detector readings will get smaller as we move closer to

x=+30cm. Since the fractional standard deviation is inversely proportional to the mean detector readings, the fractional standard deviation will increase as we progress along the +x axis.

To address the second point, it is important to realize that as we increase the number of source neutrons (statistically speaking, the sample size), the standard deviation will decrease since it is inversely proportional to the sample size. Furthermore, since the fractional standard deviation is proportional to the standard deviation, it will decrease as well. This fact is illustrated in Figure 16.

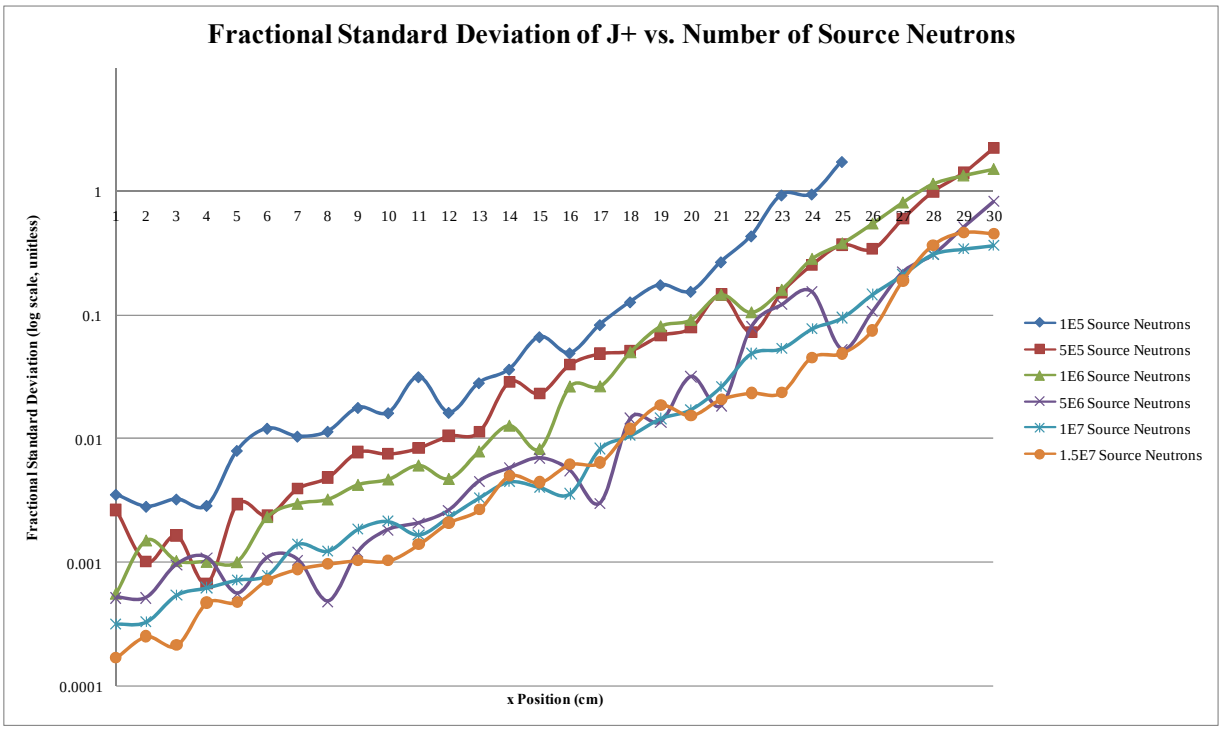


**Figure 16** ***Effect of number of source neutrons on fractional standard deviation***

Figure 16 captures the effect of the number of source neutrons on the fractional standard deviation. For example, if we compare the curve corresponding to 1E5 source neutrons to that of 1.5E7 source neutrons, we see that the fractional standard deviation markedly decreases.

*It is also of note that in Figures 9 and 16, no data is registered in the range of x=+26cm to x=+30cm (inclusive). The reason for this is that, as per the fact that a small neutron source is being simulated ($10^5$) to generate these curves, no neutrons survived into the abovementioned range. Therefore, the mean detector readings are zero and the fractional standard deviation is undefined for this particular vicinity of the +x axis.*

## 7.0 Parallel Computation Results and Discussion

The purpose of this section is to document the parallel computing results from the code. These results are namely the speedup, computing time, and figure of merit as a function of both the number of processors used and the number of source neutrons.

### *7.1 Results for Speedup*

The results for speedup are illustrated below, in Figures 17-22.

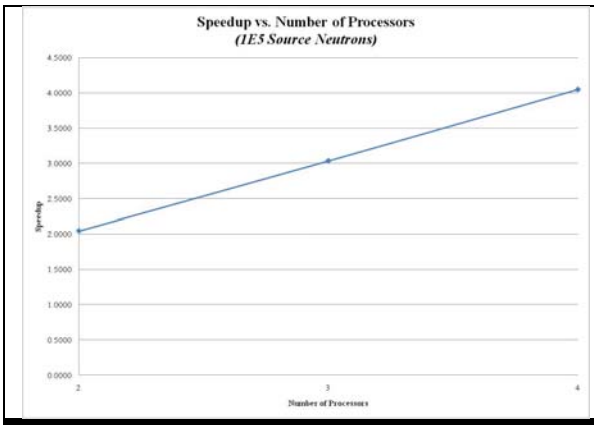


**Figure 17** ***Speedup as a function of number of processors for 1E5 Source Neutrons***

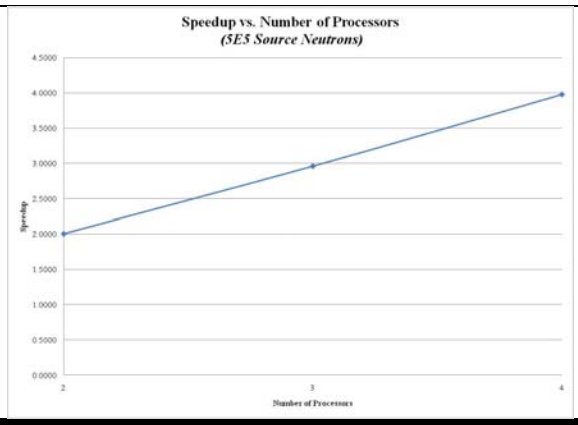


**Figure 18** ***Speedup as a function of number of processors for 5E5 Source Neutrons***

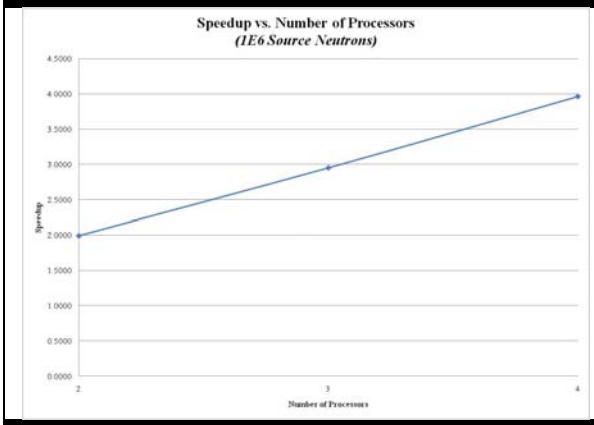


**Figure 19** ***Speedup as a function of number of processors for 1E6 Source Neutrons***

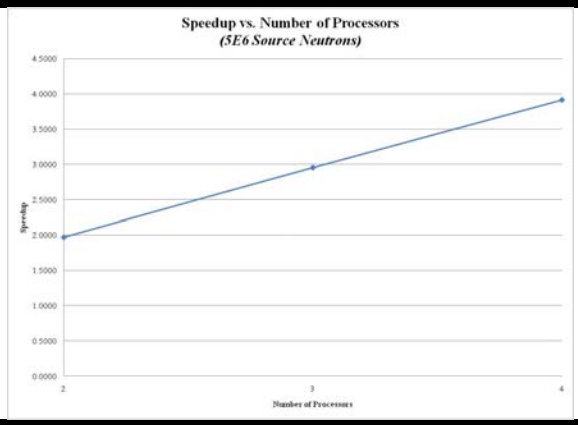


**Figure 20** ***Speedup as a function of number of processors for 5E6 Source Neutrons***

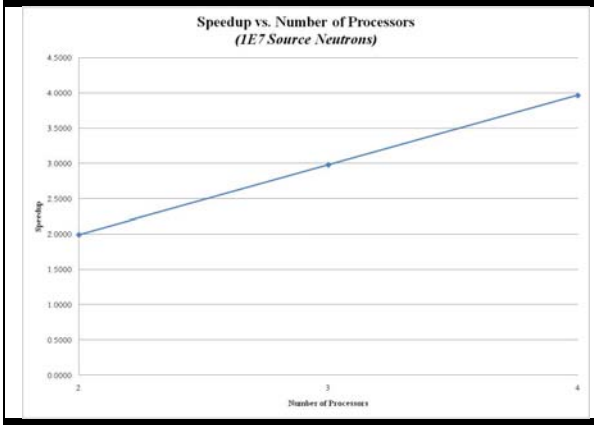


**Figure 21** ***Speedup as a function of number of processors for 1E7 Source Neutrons***

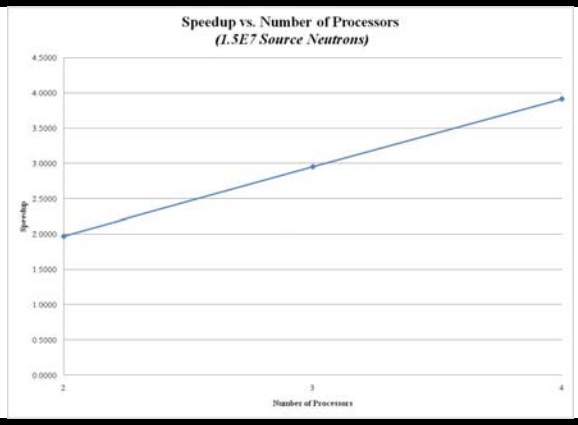


**Figure 22** ***Speedup as a function of number of processors for 1.5E7 Source Neutrons***

Figures 17-22 display the same trend which is namely the speedup linearly increases as the number of processors increase. In fact, each of the data points shown in these figures indicates that the speedup is approximately equal to the number of processors used. Thus, linear speedup has been achieved.

### *7.2 Effect of Number of Source Neutrons on Computation Time*

It is of interest to find a correlation between the number of source neutrons and computation time. What is sought after is the answer to this question: If we increase the number of source neutrons by a factor **x**, will the computation time also increase by a factor **x**?

To answer this question, we need to have a set of reference data that we can compare other data sets with. As stated in section 3.0, six different neutron source strengths were employed for the simulations with $10^5$ source neutrons being the lowest (it is of note that 1 to 4 processors (inclusive) were used to run each set of source neutrons). For the purpose of this investigation, the computation times for 1 to 4 processors to simulate $10^5$ source neutrons will serve as the reference data. Table 1 shows the results of this analysis.

| Ratio of Computation Time with reference to 1E5 Source Neutron Data | | | | | |
|---|---|---|---|---|---|
| **Number of Processors** | **Number of Source Neutrons** | | | | |
| | **5.00E+05** | **1.00E+06** | **5.00E+06** | **1.00E+07** | **1.50E+07** |
| **1** | *4.90* | *9.78* | *48.88* | *98.57* | *146.34* |
| **2** | *5.00* | *10.04* | *50.64* | *101.17* | *151.79* |
| **3** | *5.03* | *10.06* | *50.23* | *100.44* | *150.48* |
| **4** | *4.99* | *9.99* | *50.59* | *100.65* | *151.43* |

**Table 1** ***Effect of Number of Source Neutrons on Computation Time***

The data shown in Table 1 illustrate that if we are to increase the number of source neutrons by a factor **x**, the computation time will also increase by a factor of approximately **x**. For example, if the number of source neutrons increase by a factor of 5, so too will the computation time, and so on.

### *7.3 Results for Figure of Merit*

As stated in section 3.1, the Figure of Merit is a metric that allows one to judge the performance of the parallelized code. It is desired that as the number of processors increase, so too will the Figure of Merit. To investigate how the FORTRAN code behaves with respect to the Figure of Merit, a simulation was conducted using a sufficiently high neutron source strength. As a result, $10^7$ Source Neutrons were used. Figure 23 shows the effect of the number of processors used on the Figure of Merit.

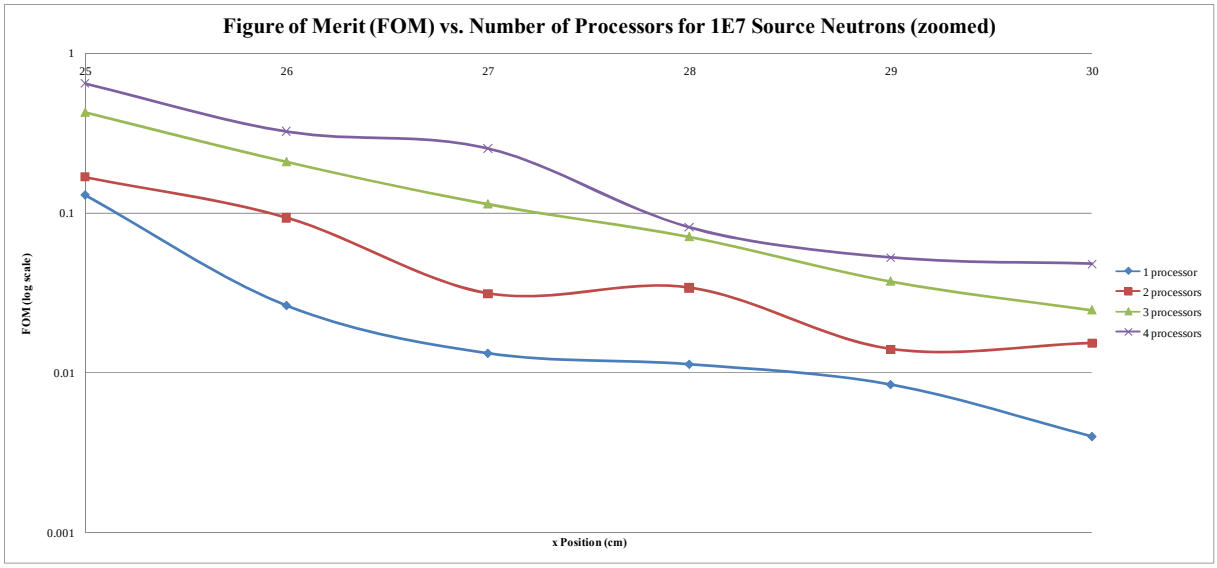


**Figure 23** ***Effect of number of processors used on the Figure of Merit***

Figure 23 shows that the Figure of Merit increases as the number of processors used increases. It is important to note, however, that this plot pertains to the right-most portion of the $D_2O$ slab (as illustrated in Figure 2). The reason for "zooming" into this region of the +x axis is simply due to the fact, as mentioned in section 6.0, the fractional standard deviation is very sensitive in this region. This is attributed to the nature of the simulation since, as a consequence of approximating neutron transport using a Monte Carlo Random Walk methodology, varying amounts of neutrons will survive into the right-most portion of the $D_2O$ slab, as we perform different runs of the FORTRAN code. *Indeed, the number of neutrons that penetrate each point along the +x axis will vary from run to run but this variation is more pronounced as we enter the latter portion of the $D_2O$ slab since neutrons will typically approach thermal energy in this vicinity*. As such, the detector readings will be different for each run. Therefore, the fact that the fractional standard deviation is dependent on the detector readings illustrates its sensitivity, and since the Figure of Merit is a function of the fractional standard deviation, it too will be sensitive in the abovementioned region of the slab.

The figure above, however, illustrates that in light of the sensitivities outlined, the simulation still yields a higher Figure of Merit as more processors are employed. This is therefore indicative of the robustness of the simulation implemented in the FORTRAN code.

## 8.0 Conclusion

The purpose and motive of this project was to implement a parallelized Monte Carlo-based neutron random walk simulation in Heavy Water ($D_2O$). From a physics perspective, this project allows one to understand how the neutron population is distributed in a material in a 2D plane. From a parallel programming perspective, an understanding of (a) how to parallelize a neutron random walk code and (b) the effect of increased source strength on the parallel computing time and fractional standard deviation. With respect to the latter point, the importance lies in the fact that typical neutron source strengths in a nuclear reactor are very large (up to $10^{14}$) and is thus difficult to simulate in a serial environment. Hence, the parallel code

developed will allow for faster simulation of large number of source neutrons (as witnessed in the results documented in this report).

## 9.0 References

[1]Ed Waller, "Transport Theory: Module II – Monte Carlo," 5 Mar. 2008, Presented at a MCSC6160 lecture at UOIT.

[2]Los Alamos National Laboratory. "ENDF-B/VI." Monte Carlo N-Particle Extended (MCNPX): Version 2.6.e. Los Alamos, NM: Los Alamos National Laboratory, 2007.

[3]Aoyama, Yukiya, and Jun Nakano, "Block Distribution," RS/6000 SP: Practical MPI Programming (Texas: IBM Corporation, International Technical Support Organization, 1999), 54-55.

[4]Lamarsh, John R., and Anthony J. Baratta, "Neutron Attenuation," Introduction to Nuclear Engineering, 3rd ed. (New Jersey: Prentice Hall, 2001), 57-60.