Simulating Neutron Transport: A Monte Carlo Approach in Serial and Parallel Environments

Agit Yesiloz
Department of Computing Sciences
Coastal Carolina University
Conway, SC, USA
ayesiloz@coastal.edu

***Abstract*—The research project examines the Monte Carlo simulation for neutron transport considering how neutrons are either reflected, absorbed or transmitted as they pass through different materials. We investigate on the interaction of neutrons using both serial and parallel computational approaches under various conditions as specified by particular values of the absorption coefficient (A), interaction parameter (C), and thickness of material (H). Our objective is to study the statistical distribution and kinetics of neutron transportation. In order to execute parallel computations, we ran our simulations on multiple processing nodes of Expanse portal at San Diego Supercomputer Centre via Message Passing Interface (MPI) framework. Thus large scale simulations become manageable since we distribute workloads across many processors greatly improving both speed and scalability of computations. To gather detailed data for analysis extensive parameter sweeps were performed with variations in A, C, and H leading to neutronics behavior running under different settings. The findings are visualized in 3D plots and 2D heatmaps that point out unique patterns which come about from interactions amongst these parameters thus providing insight into shielding efficiency and characterization of materials.**

The benefits of employing distributed computing in nuclear physics simulations are illustrated by performance benchmarks of parallel implementation, which show how high-performance computing resources can improve simulations. It not only expands our theoretical understanding of neutron transport but also paves the way for future studies that will seek to optimize and forecast neutron behaviors in complicated nuclear situations.

***Index Terms*—Monte Carlo simulation, Neutron transport, Serial and parallel computation, Message Passing Interface (MPI)**

I. Introduction

In nuclear physics, neutron transport simulations are crucial in understanding how neutrons are affected by different materials. For example, they play a major role in the design of nuclear reactors, shielding from radiations and studying fundamental nuclear interactions. In this project neutron transport is modeled using Monte Carlo methods; a statistical method that relies on random sampling for solving physical problems. Thus it makes possible to study (with the help of Monte Carlo simulations) the behavior of neutrons through diverse media.

The research has concentrated on two computational methods: serial and parallel computing. Serial computing provides a framework against which other approaches may be compared while parallel processing uses Message Pass Interface (MPI) to distribute tasks among more than one processor in use. This division is important when dealing with large scale simulations that are computationally intensive especially in determining how various physical parameters affect neutron's behaviour.

The study will set up the basic instrumental parameters such as interaction parameter of C, absorption parameter of A, and the thickness (H). For any of the active parameters, the function of the case is to decide if the absorption, reflection, or transmission takes place in the material. by assessing the parameter interaction (for example, impurity, particle size, fluence, and temperature), it identifies the material modulation in response to neutron radiation. It is an integration of these phenomena that makes it useful in material design and safety in nuclear engineering. Using a host of resources at the Expansion gateway port at the Compute center of San Diego Supercomputer, the research improves computing skills that add the ability to perform multi-parameter won-won space sweeping and perform analysis that involves data. The availability of this capacity serves the extensively researching type of interactions between A, C and H. Such research consequently leads to the more deep and accurate sense of both neutron characteristics as well as the material properties in different circumstances. In a more precise way , the project that the paper will build could be the theoretical entity of the neutron migration, but it also creates a strong base in the fields of nuclear science and engineering development for future studies. Through the use of detailed simulations and an evaluation of performance that should comprise different system configurations, the objective may be achieved that would ensure a more safe and effective system.

A. What is Neutron Transport?

The Neutron Transport is the study of the behavioral characteristics of neutrons in various materials as they move in them. This ability is vital for many areas of use, such as nuclear reactors, nondestructive testing, medical imaging and treatment. In other words, the definition of neutron transport can be seen as neutrons encounter atomic nuclei and either scattered, absorbed or fission depending on the material and energy variables. These interactions are then described by the so-called neutron transport equation which is a type of

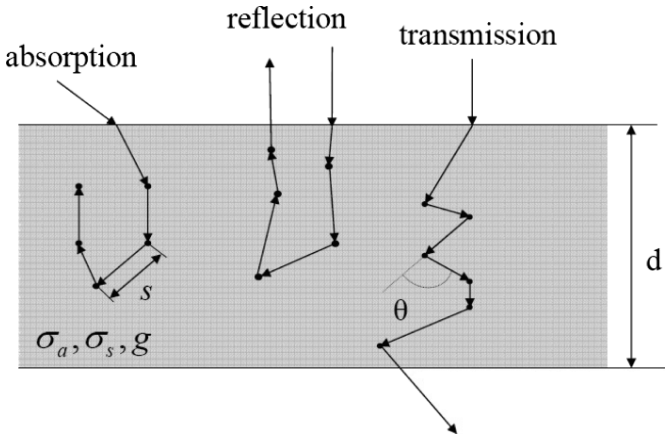

Fig. 1. Illustration of Neutron Transport. Neutrons interacting with a material are either absorbed, reflected, or transmitted. [1]

integro-differential equation which is a probabilistic approach for different of such interactions.interactions. [2]

B. Why Use Monte Carlo Methods for Neutron Transport?

In terms of complex systems, such as when neutrons bounce randomly or get absorbed, these simulations are really great because they are not tidy and not suitable for regular math. This is a tool that enables to go under the covers of what is happening in this case.Monte Carlo methods are amazingly good at visualizing neutron motion through substances, especially in complicated scenarios. Basically, they resort to random sampling to predict where neutrons will go; it matters since neutron-related phenomena are all about probabilities and randomness due to quantum mechanics. Hence, these calculations save the day when confronted with situations in which Mathematics becomes too much for conventional approaches like scattering or absorption of neutrons by chance. So, Monte Carlo simulations pull out solutions from thin air just like a magician would do with his hat while solving problems where traditional methods fail-developed ones fall short. [3]

C. What Are the Key Parameters in Neutron Transport?

In neutron transport simulations, several parameters are pivotal:

- Absorption coefficient (A): Indicates the probability per unit path length that a neutron will be absorbed.
- Interaction parameter (C): Represents the mean free path, the average distance a neutron travels between interactions.
- Material thickness (H): Affects the overall likelihood of neutrons being transmitted through, reflected from, or absorbed by the material.

These parameters determine the macroscopic behavior of neutrons and thus directly influence the design and analysis of nuclear systems.

D. How Does Serial Processing Work in Neutron Transport Simulation?

Serialized calculations are those performed on a single central processing unit, C.P.U., where each task is done in order. Despite its simplicity, it is time-consuming when applied to fast neutron transport systems with complex geometries or large numbers of neutron life histories. Validation of simulation results can be done through serial calculations which serve as baseline so that any error is detected before considering more computationally intensive parallel simulations. [4]

E. What Advantages Does Parallel Computation Offer?

Neutron transport simulations are typically carried out using parallel computing methods, whereby a large number of processors are used and the workload is divided among all the available processors. Using parallel computing reduces computation times and allows us to carry out larger and more detailed simulations before becoming computationally prohibitive. Tasks are distributed and controlled using the MPI protocol (Message Passing Interface) which has been designed to use computing resources in the most efficient way. It also enables analysis in real time, as well as larger parameter sweeps.. [5]

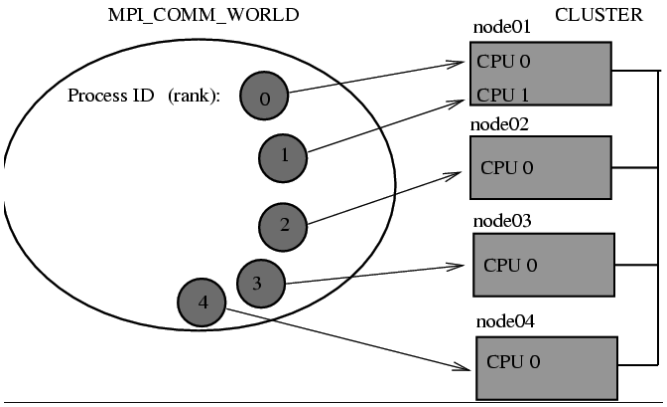

Fig. 2. Illustration of Neutron Transport. Neutrons interacting with a material are either absorbed, reflected, or transmitted. [6]

F. How Is Performance Evaluated in Neutron Transport Simulations?

Neutron transport simulation performance is traditionally assessed in terms of speedup, efficiency and scalability. Speedup is a gauge of how much faster a target parallel program runs on a given platform compared with a serial (programmed sequentially with no concurrency) version of the same code running on the same platform. Efficiency is an evaluation of how well the parallel system utilises the computational resources that are assigned to it. Scalability is an appraisal of how well a simulation manages to improve its performance as more and more computational resources are assigned to it.

G. MPI (Message Passing Interface)

The Message Passing Interface, or MPI for short, is essentially the established standard for setting up communication between parallel processes. It calls the shots, as it were, during the distribution of programs across several processors the go-to set of rules for ushering scientific programs from shared memory machines to clusters, meshes and grids. It's C, C++, Fortran or whatever else you're using to code – MPI provides a set of functions that make it natural for different processors to speak to each other in a community where everyone knows the rules and your code can live openly and reproducibly, combined with codes from all over the world, making it possible to build next-generation applications that can run on thousands or eventually millions of processors. Parallel computing would not have been able to scale up to the extent it has. [7]

II. Design

Difference of neutron transport simulation, a background of our project using parallel computing process to increase basic understandings and efficiencies of neutron interaction within the material. Simulation code using the benefits of parallel computing through MPI (Message Passing Interface), an interprocess communication system for distributed memory programming that utilises multiple processors over a single node on a parallel computational system. Unlikle of OpenMP, MPI allows processes to be executed parallelly rather than shared memory applications. Although single computer clusters could be sharing the amount of memory, various node systems are able to communicates, traverse amongst nodes, and function distinctly. For our purpose, using the Expanse portal at Centre for Information Technology Research in the Interest of Society (CITRIS) at the San Diego Supercomputer Center does not require expensive physical hardware systems. We coded our program in C language, which has a high level of computational efficiency and more precision of operation over hardware resources. To take full benefit of parallel computation nodes, specifically in this neutron transport problem, storing data and results on the hard drive is tremendously time consuming and would lead to cumbersome sequence of mass data transmission. Hence, in this section, we hope to undercover background application of data processing and operate in such a manner to distribute the worksplit among several nodes efficacious and compact. By the end of this section, we will be able to foresee an P-code used for parallel programming of neutron transport simulation, how we have implemented it, and work on optimisation strategy.

A. Technology Stack

In our project, we employed a combination of technologies to develop and analyze our parallel matrix multiplication implementation. Here's an overview:

- C Programming Language: We chose C for its performance and suitability for systems programming. It provided us with the low-level control necessary for optimizing parallel computations.
- Utilized MPI to manage communication between distributed processes, essential for our work on the supercomputing cluster.
- Visual Studio Code (VSCODE): For our development environment, we utilized Visual Studio Code, a lightweight yet powerful source code editor. It provided a user-friendly interface for writing and debugging C code.
- Windows Subsystem for Linux (WSL): To run our code in a Linux environment on Windows, we utilized Windows Subsystem for Linux. This allowed us to seamlessly work with Linux-based tools and libraries.
- Python: We used Python for data analysis and visualization. With libraries like Matplotlib and NumPy, we generated graphs to analyze the performance of our parallel matrix multiplication implementation.
- Used San Diego Super Computer Center expanse portal to submit our jobs.

1) Neutron-Transport Serial Version: This C program simulates neutron transport using the Monte Carlo method.

```
       int opt;
while((opt=getopt(argc,argv,"A:C:H:n:"))!=-1) {
switch(opt){
    case 'A':
        A = atof(optarg);
        break;
    case 'C':
        C = atof(optarg);
        break;
    case 'H':
        H = atof(optarg);
        break;
    case 'n':
        n = atoi(optarg);
        break;
    default:
        fprintf(stderr, "Usage: %s -A -C -H  -n
        \n",argv[0]);
        exit(EXIT_FAILURE);
    }
}
```

Parses the command-line arguments to set the simulation parameters.

- Uses getopt to handle options -A, -C, -H, and -n.
- -A absorption
- -C interaction
- -H thickness
- -n number of neutrons

```
        int r = 0, b = 0, t = 0;
for (int i = 0; i < n; ++i) {
    double d = 0, x = 0, L, u;
    int a = 1;

    seed[0] = (unsigned short)time(NULL);
    seed[1] = (unsigned short)i;
    seed[2] = (unsigned short)(i * 2);

    while (a) {
        u = generate_uniform(seed);
        L = -1/C * log(u);
```

```
        x += L * cos(d);

        if (x < 0) {
            ++r;
            a = 0;
        } else if (x > H) {
            ++t;
            a = 0;
        } else if (u < A/C) {
            ++b;
            a = 0;
        } else {
            d = generate_uniform(seed) * M_PI;
        }
    }
}
```

This part above simulates the transport of neutrons within the material and to determine whether each neutron is:

- Reflected
- Absorbed
- Transmitted

B. Neutron-Transport Parallel Version Using MPI

This program is designed to simulate neutron transport using parallel computing methods that faciliated by MPI. Using this code with parallezation will give us to do larger computations in shorter amount of times. This program also has similar parameter handling as serial program.

```
MPI_Bcast(&A, 1, MPI_DOUBLE, 0,MPI_COMM_WORLD);
MPI_Bcast(&C, 1, MPI_DOUBLE, 0,MPI_COMM_WORLD);
MPI_Bcast(&H, 1, MPI_DOUBLE, 0,MPI_COMM_WORLD);
MPI_Bcast(&n, 1, MPI_INT, 0,MPI_COMM_WORLD);
```

- Broadcast the simulation parameters from the root process to all processes

```
// Local simulation loop
for (int i = 0; i < local_n; ++i) {
    double d = 0, x = 0, L, u;
    int a = 1;

    while (a) {
// Generate a random number using erand48()
        u = erand48(seedp);
        L = -1/C * log(u);
        x += L * cos(d);

        if (x < 0) {
            ++local_r;
            a = 0;
        } else if (x > H) {
            ++local_t;
            a = 0;
        } else if (u < A/C) {
            ++local_b;
            a = 0;
        } else {
// Update the direction using erand48()
            d = erand48(seedp) * M_PI;
        }
    }
}
```

- The program randomly decides how far a neutron will travel before it interacts with something by using a special math function with a random number.
- It then updates where the neutron is based on how far it's supposed to travel, taking into account its direction.
- The program checks if the neutron has left the material, got absorbed, or needs a new direction
  - If it goes past the edge, it's counted as transmitted.
  - If it doesn't go far enough, it might get absorbed.
  - If neither, it bounces around inside a bit more.
- If the neutron is still in the material and hasn't been absorbed, it gets a new random direction and the process repeats.

```
// Reduce the local counts to the total counts
// on the root process
int total_r, total_b, total_t;
MPI_Reduce(&local_r, &total_r, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce(&local_b, &total_b, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce(&local_t, &total_t, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
```

- At the end of the simulation, each processor has its own count of how many neutrons were reflected, absorbed, and transmitted.
- The program adds up all these counts from every processor to get the total numbers. It uses a special MPI function called MPI Reduce that pulls all these local results and calculates the total for each type of interaction.
- All these total counts are sent to one main processor, known as the root processor. This is where the final results are gathered and can be looked at or used for further analysis.

C. Gather Sweep Data Python Script

This Python script is designed to collect the data from neutron transport simulations accross from different parameters. It effectively run the experiments on various combinations of absorption coefficient and material thickness. The scripts checks for user input to required to name an output file, and record s how many neutrons are reflected, absorbed, and transmitted. Results from the simulation are calculated with respect to their fractions. And results saved into a CSV file for easier analysis.

```
# Run the simulation and get the output
command = ['./nt-serial', '-A', str(A),
  '-C', '10.0', '-H', str(H), '-n', '1000']
result = subprocess.run(command,
   capture_output=True, text=True)
output = result.stdout

# Check if the run was successful
if result.returncode != 0:
    print(f"Error running simulation with A={A}
```

```
        and H={H}: {result.stderr}")
    continue

# extract r,b,and t values
r, b, t = parse_output(output)

if r is not None and b is not
    None and t is not None:
    # Calculate fractions
    r_n = r / 1000.0
    b_n = b / 1000.0
    t_n = t / 1000.0
    # Write the data to the CSV
    writer.writerow([A, H, r_n, b_n, t_n])
else:
    print(f"Failed to parse output
        for A={A}, H={H}")
```

D. Gather Performance Data Python Script

This is a script for the serial execution of a neutron transport simulation when evaluating the performance: It runs the exercise for different numbers of neutrons (N values) and different numbers of processors to evaluate the scaling of the performance. The script records the time it took to run each simulation for N (number of neutrons) with a serial version of the program and with its parallel version using the MPI library for different numbers of processors that is used. The results of all the experiments, number of processors used and the execution times for the serial and parallel runs are stored in to a CSV file.

```
# Run the serial program
start_time = time.time()
subprocess.run(['./nt-serial', '--A', str(A),
'--C', str(C), '--H', str(H), '--n', str(N)])
end_time = time.time()
time_serial = end_time - start_time

# Loop over the number of processes
for p in processes:
    # Run the MPI program
    start_time = time.time()
    subprocess.run(['mpirun', '-np', str(p),
    './nt-parallel', '--A', str(A), '--C',
str(C), '--H', str(H), '--n', str(N)])
    end_time = time.time()
    time_MPI = end_time - start_time

# Write the data to the file
writer.writerow([N, p, time_serial, time_MPI])
```

III. Experimentation and Results

float This section elaborates on the experimental setup, execution process, and the outcomes attained within the Expanse portal, a high-performance computing environment. Following execution, the resulting output files underwent analysis via a Python script to create performance graphs.

A. Experimental Setup

The bash script intended to be submitted as a job to a high-performance computing (HPC) cluster using the Slurm workload manager. The script below designed to specify the job requirements and environment settings for running a parallelized program using MPI on an HPC cluster managed by Slurm on expanse portal.

```
#!/bin/bash

#SBATCH --job-name="neutron_sim"
#SBATCH --output="output_%j.csv"
#SBATCH --partition=shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=1
#SBATCH --mem=12GB
#SBATCH --account=ccu108
#SBATCH --export=ALL
#SBATCH -t 00:45:00
module purge
module load slurm
module load cpu
module load gcc/10.2.0
module load openmpi/4.1.1
mpicc -Wall -std=c11 -o nt-parallel
    nt-parallel.c functions.c -lm

# Define N values for the sweep
N_VALUES=(96000 144000 160000 )

# Define the values for A, C, and H
A=0.01
C=200
H=100

# Start the loop within the SLURM script
for N in "${N_VALUES[@]}"; do
    for THREADS in $(seq 1 64); do
        echo "Running simulation with N=${N}
        and THREADS=${THREADS}"
    # Use srun to specify the number of tasks
            dynamically
        mpirun -n $THREADS ./nt-parallel --A
        $A --C $C --H $H --n $N
    done
done
```

B. Execution Procedure

The execution procedure involved submitting the implemented matrix multiplication algorithms to the Expanse portal, a high-performance computing environment. The programs were compiled with the necessary dependencies, including OpenMP support for MPI, and then submitted as jobs to the cluster using the Slurm workload manager.

Once the jobs were submitted and executed, the output files containing the computed results were generated. These output files were then analyzed using a Python script designed to extract relevant performance metrics and generate graphical representations for further analysis.

IV. Results

The results obtained from the Expanse portal after executing the matrix multiplication algorithms provided valuable insights into their performance. To analyze these results effectively, a Python script leveraging libraries such as Matplotlib, NumPy, and Pandas was utilized.

A. Time

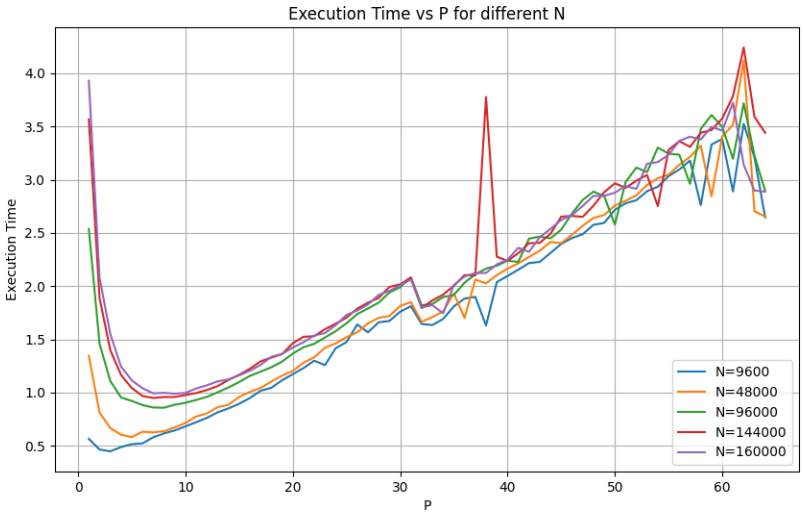

Fig. 3. Execution Time Versus Thread Count for Different Neutron Counts. The chart displays how the simulation's execution time changes with the number of threads used, underlining the effects of parallel processing on neutron transport calculations in the Expanse supercomputer framework.

As the number of processors increases, the execution time initially decreases. This trend suggests that adding more processing power substantially speeds up the calculations but the does not perform well with larger number of threads.

B. Speedup

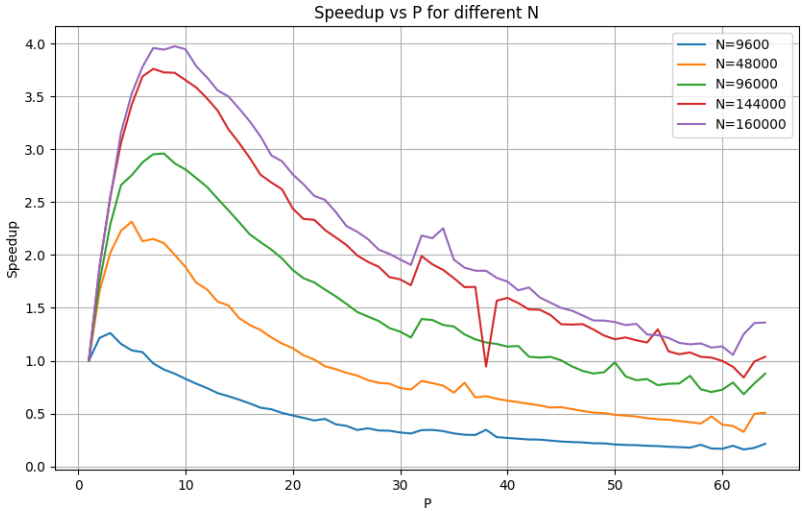

Fig. 4. This graph shows the speedup of a parallel neutron transport simulation relative to the number of threads (P) for various problem sizes (N), indicating the optimal threading point for computational efficiency in a high-performance computing environment.

Speedup in parallel computing refers to the performance gain of running a task on multiple processors in comparison to a single processor. Essentially, if a task takes less time to complete on multiple processors than it would on one, that reduction

in time is the speedup. It's calculated by taking the time a task would take on a single processor and dividing it by the time it takes using multiple processors.

$$S(P) = \frac{T_{serial}}{T_{parallel}(P)} \tag{1}$$

The graph shows that while parallelization can significantly speed up computation, there's a complex relationship between the number of threads, the problem size, and the efficiency gains, underscoring the importance of tuning parallel applications for specific problem sizes and computing environments.

C. Efficiency

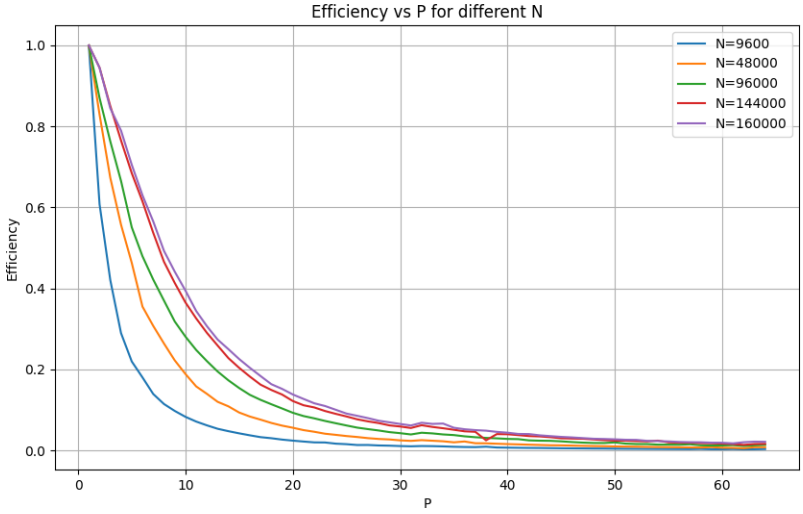

Fig. 5. Efficiency versus Thread Count for Various Neutron Counts. This plot compares the computational efficiency of a parallel neutron transport simulation across different thread counts (P) in the Expanse supercomputing environment, illustrating the trade-offs between processing power and problem size (N).

Efficiency is about how well the multiple processors are being utilized. It's calculated by taking the speedup and dividing it by the number of processors. Perfect efficiency means that all processors contribute equally and effectively to completing the task, without any waste of resources or time. In reality, efficiency often decreases as the number of processors increases due to factors like communication overhead or imbalance in the distribution of work among processors.

$$E = \frac{S}{P} = \frac{T_{serial}}{P \cdot T_{parallel}(P)} \tag{2}$$

This graph portrays how efficiently a parallelized neutron transport simulation runs as we increase the thread count. It shows that with fewer threads, the efficiency starts off high, making good use of the parallel setup. However, as more threads are added, the efficiency starts to go down, indicating that the communication between threads—begin to benefits of additional parallelism. Interestingly, the bigger the problem size (higher N values), the longer the efficiency remains high as thread count increases, suggesting that larger problems can be more effectively parallelized.

D. Neutron Transport Visualization

1) Neutron Interaction with Varying Material Thickness:

- The sharp drop in the blue line indicates that a material's ability to reflect neutrons decreases as it gets thicker; most neutrons are reflected at thin levels, but fewer are as thickness increases.
- Absorption (Grey Area): The grey area grows with material thickness, showing more neutrons are absorbed. It levels off indicating a limit to how much the material can absorb, regardless of further increases in thickness.
- Transmission (Orange Area): Transmission decreases as material thickness increases since more neutrons are blocked or absorbed. However, some neutrons still pass through, indicating that no material provides complete shielding.

2) Fig. 7:

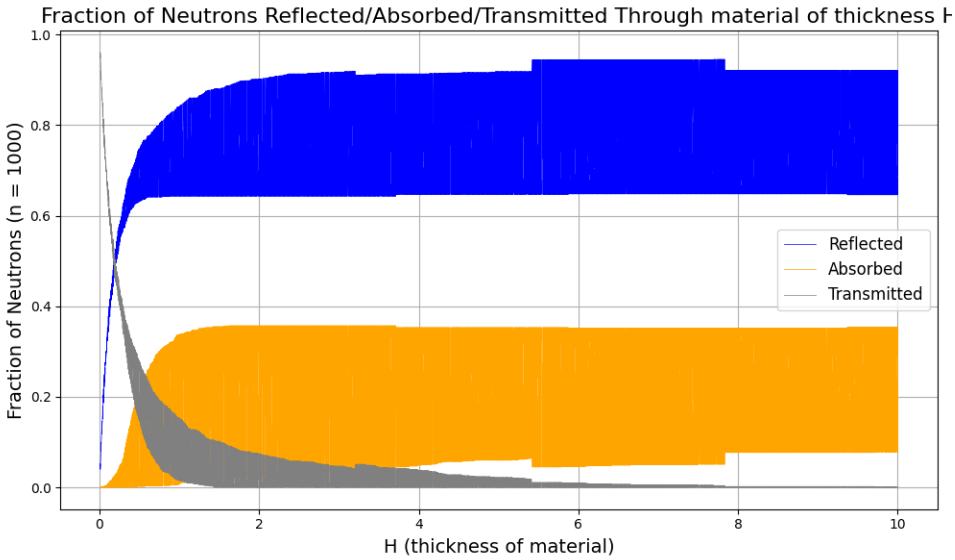

Fig. 6. This graph shows Neutron Behavior in Material by Thickness - Reflectance diminishes, absorption saturates, and transmission persistently decreases as material thickness increases.

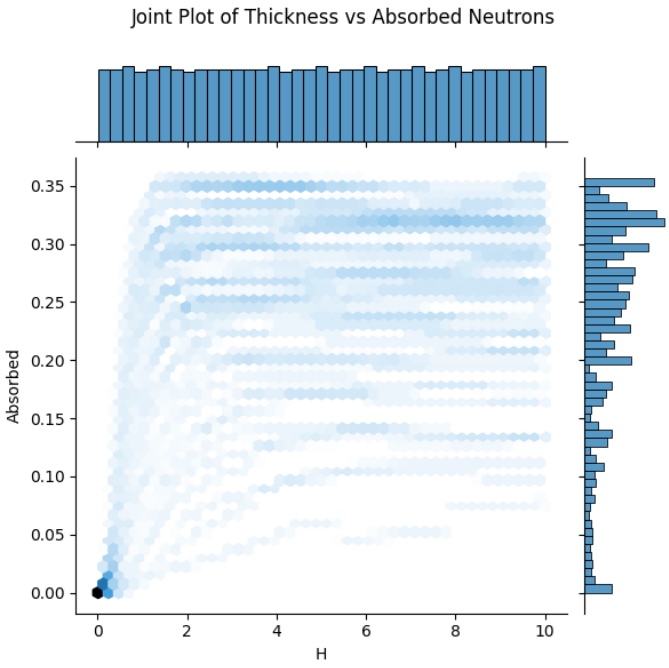

Fig. 7. The histogram displays a uniform sampling of material thicknesses, and the adjacent scatter plot illustrates an increase in neutron absorption with greater material thickness.

3) Fig. 8:

- Upper Histogram: Displays the uniform frequency of thickness measurements for neutron absorption, suggesting that each thickness from 0 to 10 units was equally examined.
- Right Scatter Plot: Illustrates that as the material thickness (H) increases, the fraction of neutrons absorbed also increases, with a high density of points at lower thicknesses indicating a greater number of observations in these regions.

4) Fig. 9:

- Upper Histogram: This histogram indicates a consistent number of measurements across all levels of material thickness for neutron reflection data.

- Right Scatter Plot: Reveals a decrease in the fraction of neutrons reflected as the thickness (H) of the material increases, with the most significant change occurring at the lower thickness levels.

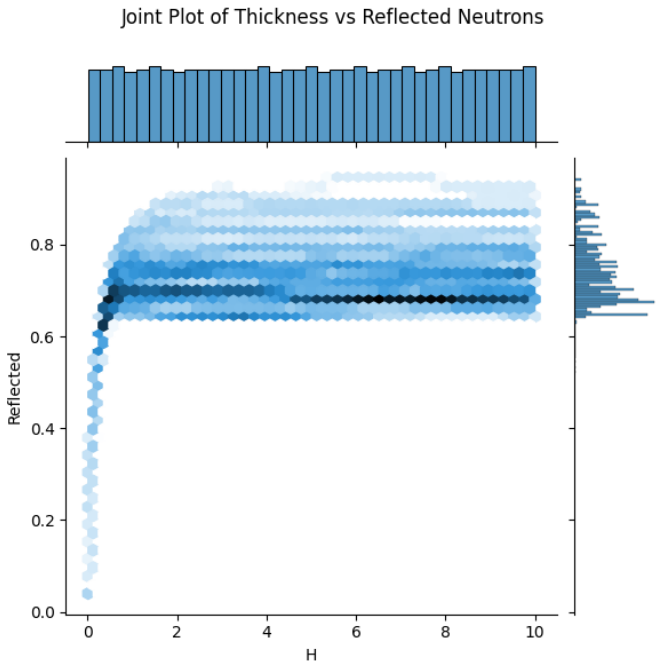

Fig. 8. Equal Measurement Frequency Across Material Thicknesses for Neutron Reflection

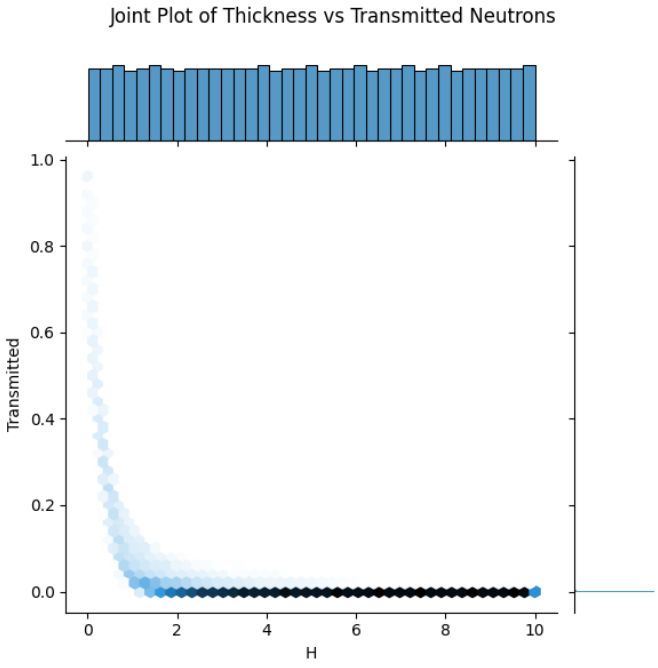

Fig. 9. This graph shows Neutron Behavior in Material by Thickness - Reflectance diminishes, absorption saturates, and transmission persistently decreases as material thickness increases.

V. Conclusions

In conclusion, this study has clearly provided us the power Monte Carlo simulations to understand nature of neutron transport with various materials. Through various experimentation in both serial and parallel computatation, we have demonstrated the outcomes that influenced by different material properties such as thickness, absorbtion coefficient, and interaction parameters.

The utilization of parallel processing by using Message Passing Interface(MPI) has proved us the significance of the computing large scale computations in parallel environments. As we can see from the graphs it provided us a faster computation times, and capability to manage more complex systems. Our findings demonstrate a clear pattern of neutron behavior decrement in reflection and transmitting with increased material thickness.

The integration of high performance computing in neutron transport simulations has proven to be an important asset for us, advance in nuclear physics and engineering field.

References

[1] D. Modric, "Monte carlo modeling of light scattering in paper," *Journal of Imaging Science and Technology*, 2009.

[2] E. E. Lewis, "Fundamentals of nuclear reactor physics," 2008.

[3] A. Haghighat, "Monte carlo methods for particle transport," 2020. [Online]. Available: https://www.researchgate.net/publication/343171994

[4] E. E. Lewis and J. W. F. Miller, "Computational methods of neutron transport," 1993.

[5] M. J. Quinn., "Parallel programming in c with mpi and openmp," 2003.

[6] spagnuolocarmine. (2024) Codinggame. [Online]. Available: https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/mpi-programming

[7] ——. (2024) Message passing interface. [Online]. Available: https://en.wikipedia.org/wiki/Message$_{P}assing_{I}nterface$