The Monte Carlo Simulation of Neutron Transport: A Comparative Analysis of Serial and Parallel Implementations

Karolin Krzeminski
Dept. of Computing Sciences
Coastal Carolina University
Conway, SC, USA
kkrzemin@coastal.edu

***Abstract*—This paper presents a comparative study of serial and parallel implementations of a Monte Carlo simulation for neutron transport in a two-dimensional model. The simulation estimates the frequency of neutron interactions, such as the reflection, absorption, or transmission as they encounter a homogeneous plate of varying thickness. Key performance metrics such as computational time, speedup and efficiency were analyzed. For the parallel computation, we used the Message Passing Interface (MPI) framework to distribute the simulation across multiple processors, aiming to enhance computational efficiency and scalability. Our findings reveal significant performance improvements in the parallel algorithm.**

I. Introduction

Monte Carlo simulations are significant in the field of neutron transport, offering insights into fundamental interactions within a given medium. The core of our simulation is a Monte Carlo method for modeling neutron transport within a two-dimensional homogeneous medium. Neutrons emitted from a source interact with a plate characterized by its thickness H and are subject to reflection, absorption, or transmission. The interactions are determined problematically, guided by the medium's cross-sectional parameters for capture and scattering.

Despite their accuracy and flexibility, Monte Carlo simulations are resource intensive, often requiring significant computational effort to produce reliable results. This computational demand grows with the complexity of the system, such as a two-dimensional model of neutron transport through a homogeneous plate with variable thickness. Such models are important in understanding how neutrons interact with materials whether they are reflected, absorbed, or transmitted.

This paper provides a comparative analysis between serial and parallel processing for the Monte Carlo simulation. Employing the Message Passing Interface (MPI), we distributed the workload across multiple processors, aiming to significantly reduce computational time while enhancing speedup and efficiency.

We evaluated computational time, speedup, and efficiency as our primary metrics. The comparative results demonstrate that our MPI parallel algorithm significantly enhances performance with a reduction in computation time but also enhances the scalability of Monte Carlo simulations for neutron transport.

II. Methods

We utilized the Message Passing Interface (MPI) to implement the algorithm in parallel. The MPI framework enabled us to distribute the computational workload across multiple processors, each running a subset of the simulation trials. This parallel approach aimed to reduce overall computational time and improve scalability for larger-scale simulations.

A. Initialization

Upon initialization, each processor is assigned a unique seed for the random number generation to ensure independence of simulation trials. This seed is derived from a combination of the current time, the processor's rank within the MPI environment, and the process ID. The master process (rank 0) parses command-line arguments to set the physical parameters of the simulation, such as the mean free path, absorption cross-section, plate thickness, and the total number of simulation trials.

Snippet 1: Initialization

```
double uniformRandom(unsigned short seed[3]) {
    return erand48(seed);
}
    time_t now = time(NULL);
    unsigned short seed[3];
    seed[0] = (now & 0xFFFF) ^ (rank & 0xFFFF);
    seed[1] = (now >> 16) ^ (rank >> 8);
    seed[2] = getpid();
```

B. Neutron Interaction Simulation

Each processor executes a local subset of the total simulation trials. During each trial, the path of a neutron is tracked until an interaction event occurs, either reflection, absorption, or transmission. The distance ***L*** a neutron travels before interacting with an atom within the plate follows an exponential distribution with a mean of 1/C, and the neutron's new direction after a scattering event is uniformly distributed between 0 and pi.

The neutrons' interactions are simulated in a loop until one of three conditions is met: the neutron is absorbed by the medium, reflected back towards the source, or transmitted through the plate. The outcomes of these interactions are accumulated locally on each processor.

Snippet 2: Neutron Interaction Simulation

```
    int local_n = n / size;
    int r = 0, b = 0, t = 0;

    // Simulation
    for (int i = 0; i < local_n; ++i) {
        double d = 0;
        double x = 0;
        int a = 1; // true

        while (a) {
            double L = -1 / C
            * log(uniformRandom(seed));
            double u = uniformRandom(seed);
            x += L * cos(d);

            if (x < 0) {
                // Reflected
                ++r;
                a = 0; // false
            } else if (x > H) {
                // Transmitted
                ++t;
                a = 0; // false
            } else if (u < Cc / C) {
                // Absorbed
                ++b;
                a = 0; // false
            } else {
                //new direction
                d = u * M_PI;
            }
        }
    }
```

C. Data Reduction and Time Measurement

After completing the simulations, the local counts of reflected, absorbed, and transmitted neutrons are reduced across all processors using MPI's reduce function. The master process gathers these results to calculate the global totals for each interaction type. To measure the efficiency and performance of the simulation, the wall-clock time from the start to the end of the entire simulation process is recorded on the master process, providing a metric for the parallel algorithm's speedup over the serial implementation.

Snippet 3: MPI$_{R}educe$

```
// Collect results using MPI_Reduce
    int total_r, total_b, total_t;
    MPI_Reduce(&r, &total_r, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Reduce(&b, &total_b, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Reduce(&t, &total_t, 1, MPI_INT,
    MPI_SUM, 0, MPI_COMM_WORLD);
```

Snippet 4: Wall Time

```
MPI_Barrier(MPI_COMM_WORLD);
    double endOverall = MPI_Wtime();

    // Calculate the elapsed time
    double elapsedOverall = endOverall
    - startOverall;
```

D. Output

The master process calculates the normalized frequencies of each interaction type by dividing the total counts by the number of trials *n* and prints the results. Additionally, it outputs the total elapsed time for the parallel execution of the simulation, giving insight into the parallelization benefits over a single-process execution.

Performance

Performance was measured by comparing the execution time of the Monte Carlo simulation in terms of overall time. These measurements were taken under different conditions by varying the number of threads and the size of the computation values. The specific metrics used to evaluate performance include:

- **Speedup**: Calculated by dividing the execution time of the serial program by the execution time of the parallel program for overall, work, and I/O times. The formula used for calculation is:

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

- **Efficiency**: Calculated as the speedup divided by the number of threads (p), providing insight into how effectively the parallel resources are utilized. The efficiency formula is:

$$E = \frac{S}{P}$$

Experimental Setup

The experiments were conducted using values of sizes 85,000,000 to 150,000,000 in 500,000 increments, with varying iterations to maintain the overall work size. The number of threads was varied across 1 to 64 to observe the impact of thread count on performance. The performance metrics were collected for each combination of value size, number of iterations and thread count.

The environment for the experiments was conducted on the Expanse supercomputer. The Expanse system is equipped with multi-core processors, providing a robust platform for executing parallel computing tasks.

By varying the conditions under which the Monte Carlo simulation was performed, we were able to analyze the effects of parallelization on computational efficiency and identify the optimal number of threads for different value sizes.

III. Data Visualization

In order to visualize the neutron transport, data generation and visualization programs were created using Python. We created a 2D plot, 3D plots, and heat maps to show the absorption, reflection and transmission of neutrons with respect to the thickness of material H. View Figure 1 - Figure 4

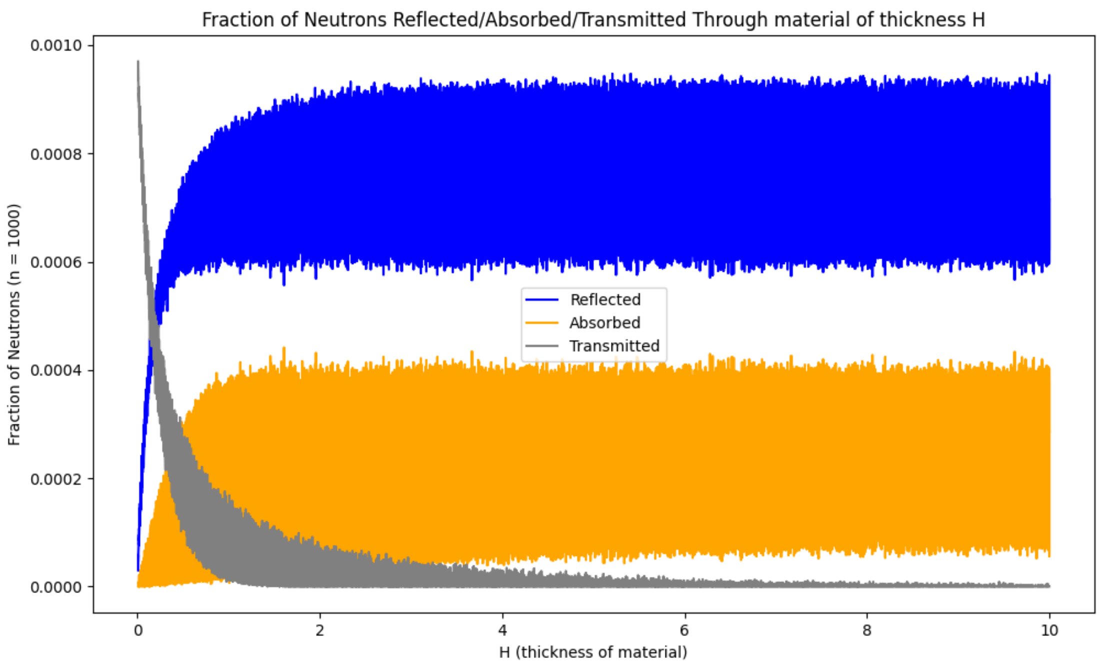

Figure 1: 2D Neutron Interaction Plot - As H (thickness of material) gets thicker, the amount of neutrons reflected grows, whereas the amount of neutrons transmitted decreases. The absorption of neutrons shows an initial increase as the material gets slightly thicker, indicated by the rising orange area.

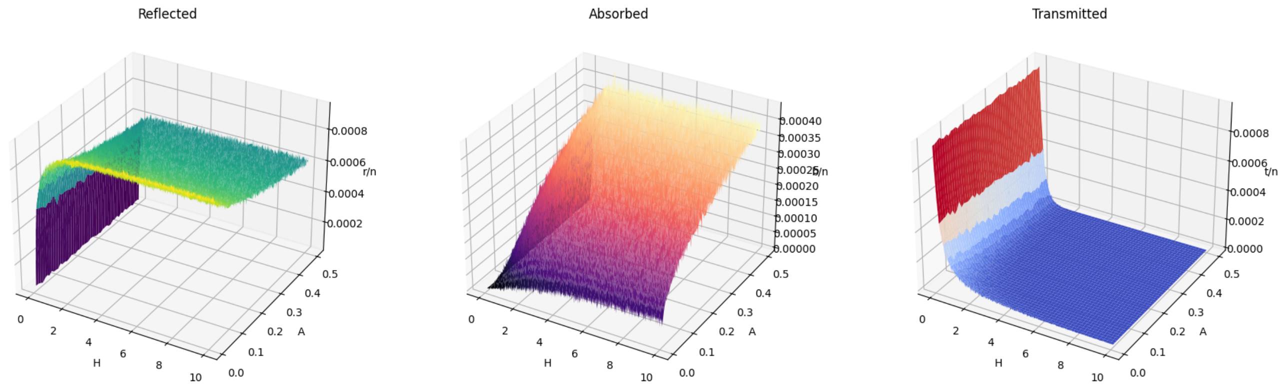

Figure 2: 3D Neutron Interaction Plot

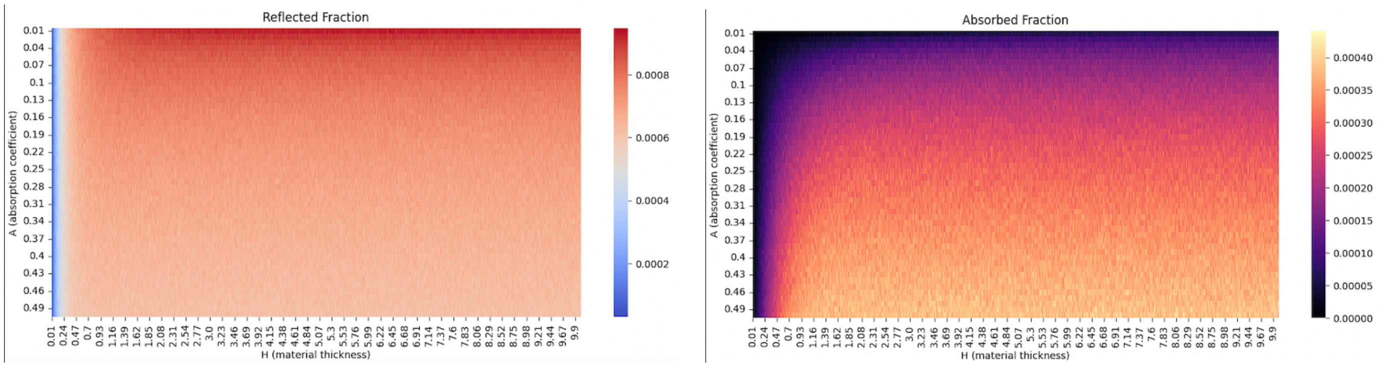

Figure 3: Reflected and Absobed - Heatmap Neutron Interaction Plots

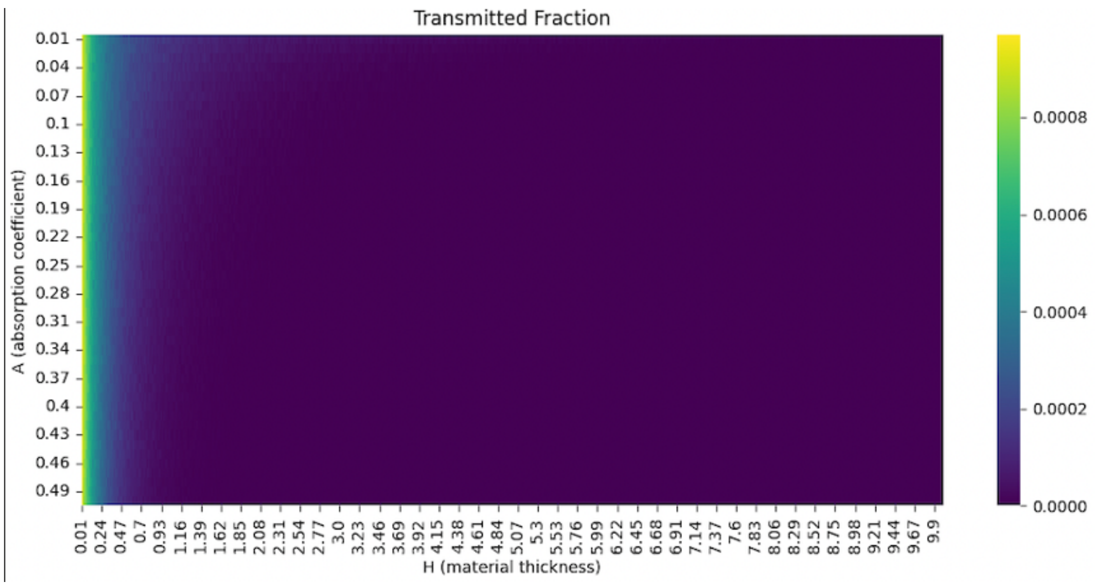

Figure 4: Transmitted Heatmap Neutron Interaction Plot

IV. Results

In order to visualize the performance of our MPI program we have created timing graphs for overall, speedup and efficiency. Refer to Figure 5 - 7.

V. Discussion

The primary objective of this study was to assess the computational gains achieved through parallel processing in the context of neutron transport simulations via the Monte Carlo method. The performance metrics for computational time, speedup, and efficiency were the benchmarks against which the serial and parallel implementations were evaluated.

The results of the simulation confirmed that parallel processing can significantly reduce computational time. Our findings showed that for the neutron transport problem, the MPI-based parallel implementation outperformed the serial one.

Furthermore, the study's data visualization revealed the interactions of neutrons with the material as the thickness varied. This demonstrated the capabilities of the Monte Carlo method in capturing the physics of neutron transport.

Contents

References

[1] W. contributors, "Monte carlo method." https://en.wikipedia.org/wiki/Monte$_Carlo_method$ 2023.

[2] P. S. Pacheco and M. Malensek, *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers, an imprint of Elsevier, 2022.

[1] [2]

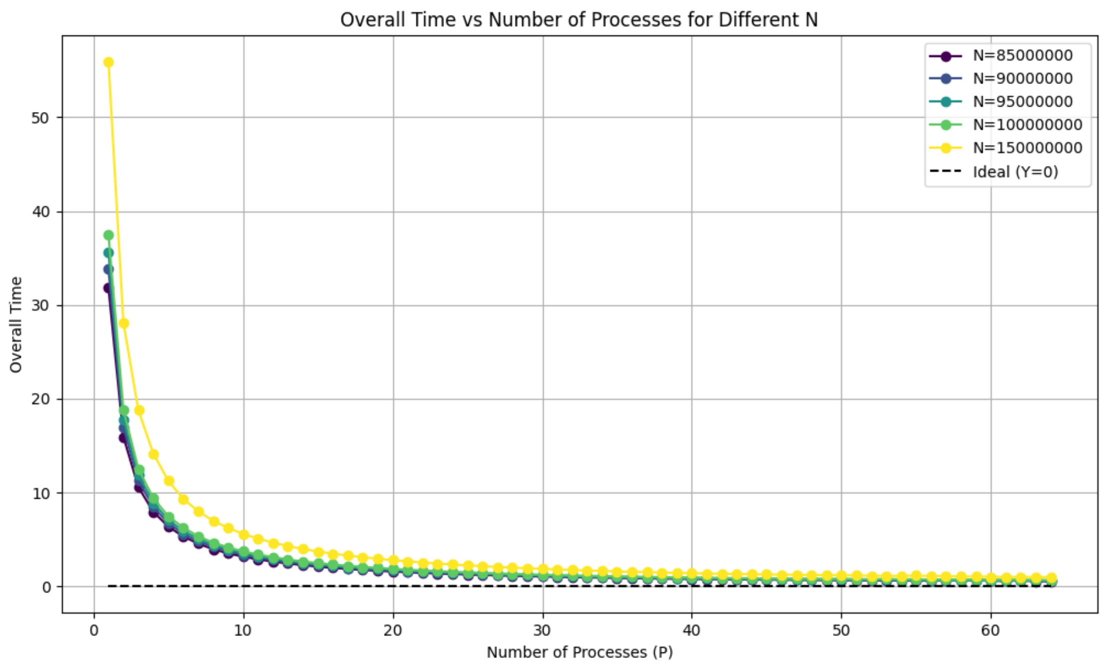

Figure 5: Overall Time - All sizes of N significantly decrease in overall time when more threads are introduced.

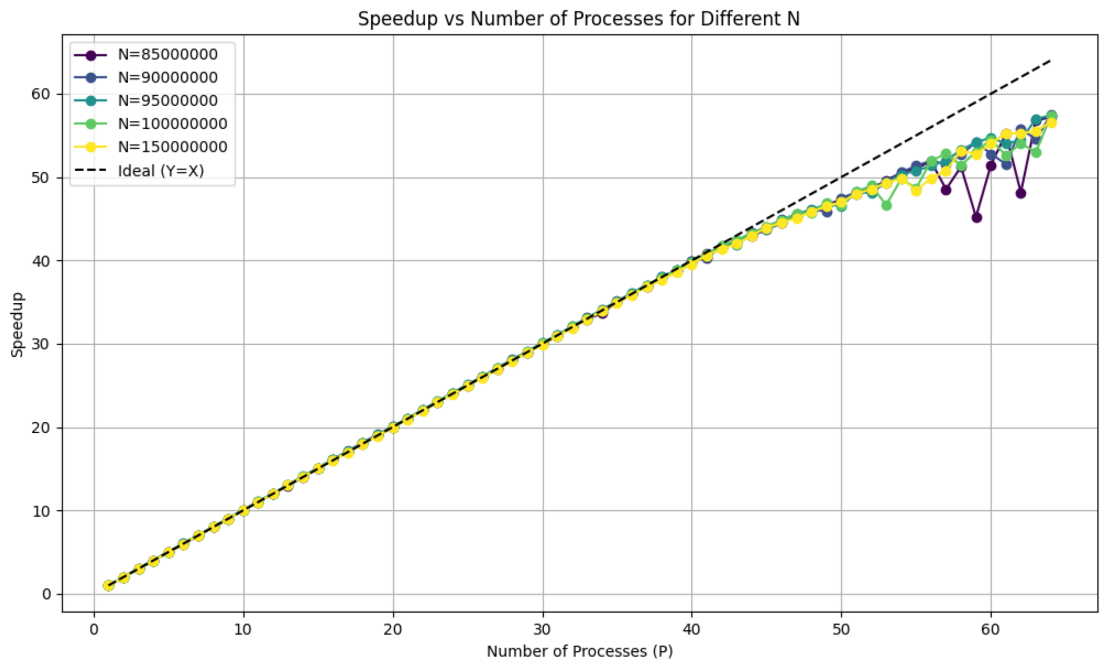

Figure 6: Speedup - For all sizes of N, a nearly linear speedup is seen up to about 45 threads, where they begin to diverge from the ideal line.

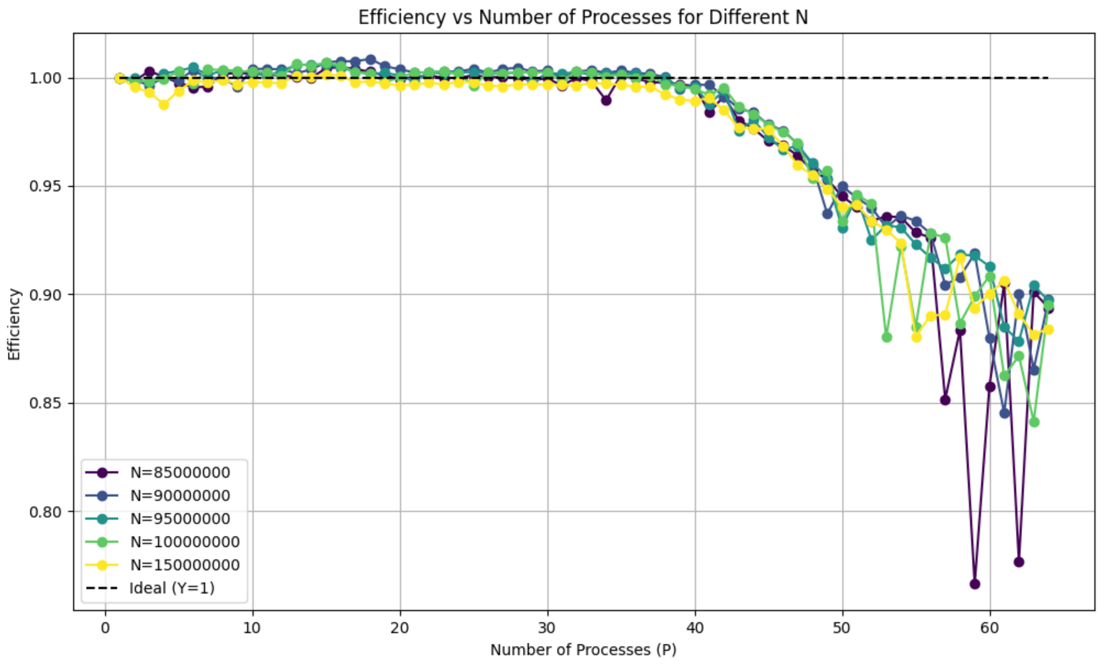

Figure 7: Efficiency - The efficiency for all sizes of N is near ideal at Y=1 until about 45 threads where they begin to diverge.