Enhancing Performance of Monte Carlo Simulation Algorithms Using Parallelization Techniques

1 Introduction

Monte Carlo simulations, a cornerstone in scientific and computational disciplines, offer a powerful means to model complex systems by employing random sampling methods. Their versatility spans physics, finance, engineering, and beyond, enabling researchers to tackle diverse problems with numerical precision. However, as datasets grow larger and analyses demand real-time insights, the need for optimizing these simulations becomes paramount. The inefficiency of serial execution generally poses a significant bottleneck in achieving timely results, prompting the exploration of parallelization techniques.

2 Algorithm Overview

2.1 Metropolis

The Metropolis algorithm is a foundational Monte Carlo method used for generating samples from a probability distribution, particularly applicable in statistical mechanics and computational physics. Here is how the algorithm operates (Chib):

1. Initialize a graph with randomized spins
2. In the graph, choose a spin at random and immediately flip it.
3. Based on the nearest neighbors of that spin, decide to accept or reject the flip
    a. If all the neighbors agree, then the flip is accepted unconditionally

   b. If all the neighbors don't agree then the flip is only accepted based on probabilistic criteria. If the flip is rejected, the spin reverts to its original state.
4. Repeat steps 2-3 for all points in the graph.

The criterion by which the algorithm decides to accept or reject the proposed change is called the decision criterion. The criterion is calculated as $e^{(P_{proposed}-P_{current})/T}$ where P represents the probability of the flip and T represents the total "Temperature" of the system. The Metropolis algorithm's simplicity and effectiveness make it indispensable, providing a method to tackle problems with numerous degrees of freedom or where analytical solutions are impractical.

2.2 Swedsen-Wang

The Swedsen-Wang algorithm when devised was quite different from the popular Metropolis Algorithm for Ising graphs. Instead of using the single-flip method employed by the Metropolis Algorithm, the Swedsen-Wang algorithm implies that every spin in the graph is directly (or indirectly) connected (Frenkel). Here is how the algorithm works:

1. For a given graph with randomized spins, choose a single spin
2. Create a "bond" with all of the nearest neighbors of that spin with a probability of k where their spins are the same.
3. Every spin connected to the original point via a "bond" is part of the same cluster.
4. The spins in that cluster are flipped with a probability of 0.5 (as there are only 2 values a spin can have)
5. Remove the bonds in the cluster and repeat the algorithm

Given how the algorithm works, it differs from the Metropolis algorithm as it flips entire clusters of points rather than individual points. The algorithm can flip entire clusters because all of the

spins within a cluster are uncorrelated to the spins in other clusters (Frenkel). Therefore, the spins in a cluster can be assigned a new spin value independently of other clusters. Also, a cluster can be any number of spins, as long as there is at least one spin in each cluster. The Swedsen-Wang algorithm is popular today due to how it does not slow down near critical points (Frenkel). This means that in places in the graph where getting to a convergence might slow due to a fluctuation, the Sweden-Wang does not experience a significant slowdown.

2.3 Wolff and Single-Cluster

The Wolff algorithm is similar in concept to the Swedsen-Wang algorithm. It can be seen as a variant of the Swedsen-Wang, notably a Single-cluster variation. While the Swedsen-Wang algorithm creates both small and large clusters, the Wolff algorithm only creates a single cluster and that cluster is always flipped (Wolff). Here is how the algorithm works:

1. Choose a single spin randomly
2. All of the nearest neighbors of that spin are added to a cluster if spins i and j are the same and if the pair of spins already hasn't been bonded
3. The spins that are added to the cluster are also added to a stack
4. For each spin in the stack, repeat steps 2-3 until the stack is empty
5. Flip the cluster once the stack is empty

In this algorithm, the cluster is always flipped making it a "rejection-free" algorithm like Swedsen-Wang (Wolff). Note that the Wolff algorithm can be simplified a bit when adding a spin to the cluster, if that spin is immediately flipped, step 5 can be removed and it protects against having the same spin added twice to the cluster. The decision to accept or reject the cluster flip is similar to the probabilistic criteria used by the Metropolis algorithm.

3 Methods

3.1 Starter Code Implementation

For the implementation initially, we were given a starter code that had a basic clustering algorithm with a spin update function that used the methodology of a heat bath. This code also outputs the size of the graph, the number of clusters, the number of spins per cluster, and the spin graph after the flipping. This code provided a good base for implementing both the Swedsen-Wang and Wolff algorithms and we used it as a template for the implementation. In addition, we also decided to use the starter code as an additional algorithm test with the other 2 algorithms.

3.2 Parallelization Implementation

We decided to parallelize the algorithms by using OpenMPI to parallelize any loops present in the algorithms. We decided to use OpenMPI as it provided a simpler method to implement parallelization and it was also easier to test on the SCC due to not needing a GPU to run the parallelization. So, we used the pragma keyword in various loops in the algorithm to parallelize those loops during execution.

3.3 Testing Method

For testing, we decided to run each algorithm with 2 sets of parameters. These parameters were the size of the grid N, and the number of threads used T. For each N, we decided to run 3 different values of T to determine if the parallelization reduced execution time. We decided to use the following values of T: 1, 4, and 8. We also had 4 values of N with a minimum value of 256 and a maximum value of 16384. We decided to use these values because a reduction in

execution time should be more apparent with a high value of N. We also calculated certain metrics for each N, namely a speed-up metric to quantify the difference between serial and parallelized runtimes in addition to an efficiency metric which calculates the value of runtime per thread used.

4 Results

4.1 Statistics and Data

In evaluating the parallel performance of Monte Carlo simulation algorithms, our study focused on the speedup and efficiency metrics across varying thread counts and grid sizes. Figures 1-3 illustrate the runtime performance of tested algorithms across different thread counts and grid sizes. Surprisingly, parallelization of the Swendsen-Wang algorithm yielded suboptimal results, often slower compared to its serial counterpart.

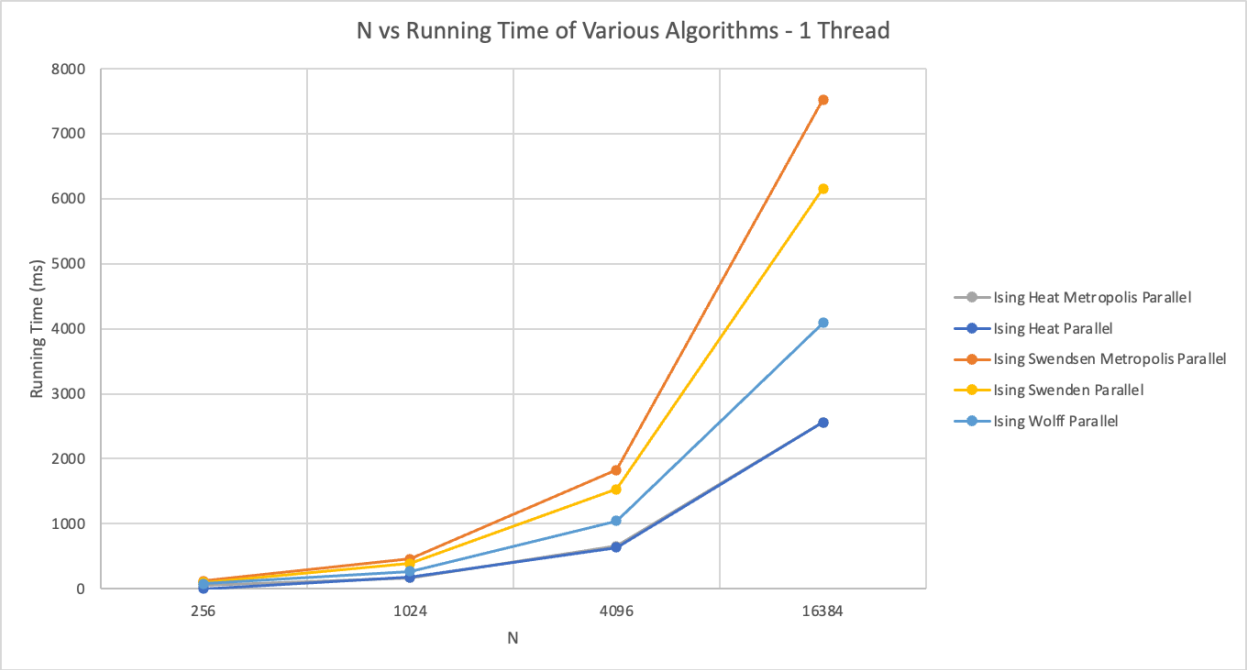

Figure 1: N vs Running Time for All Tested Algorithms - 1 Thread

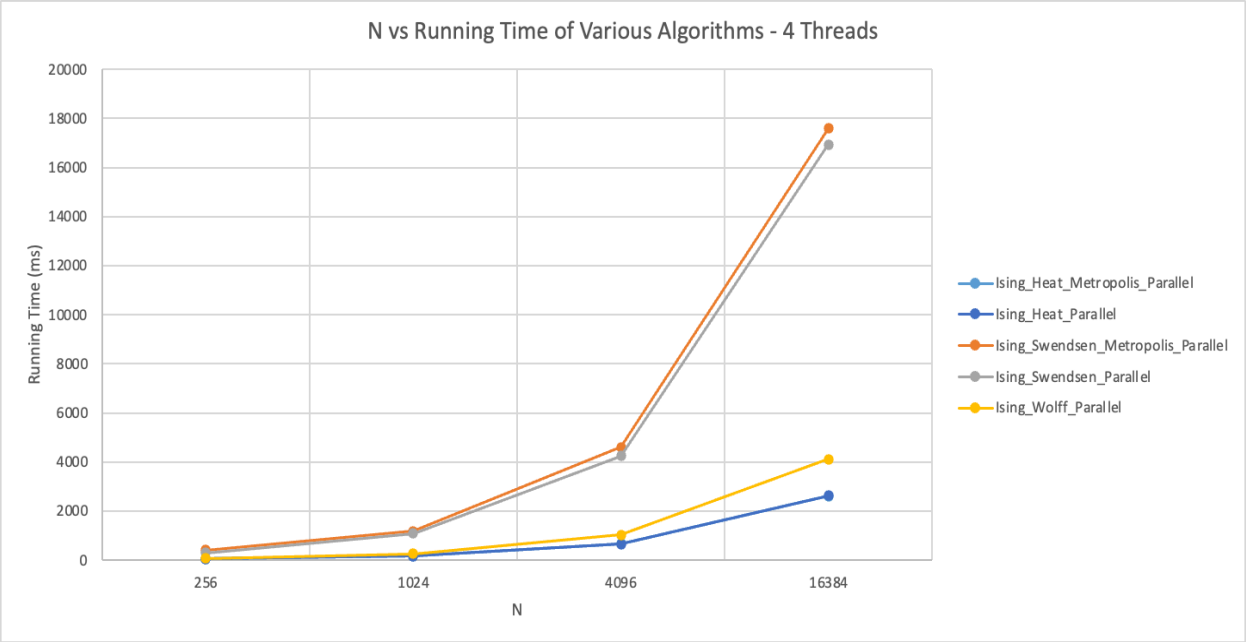

Figure 2: N vs Running Time for All Tested Algorithms - 4 Threads

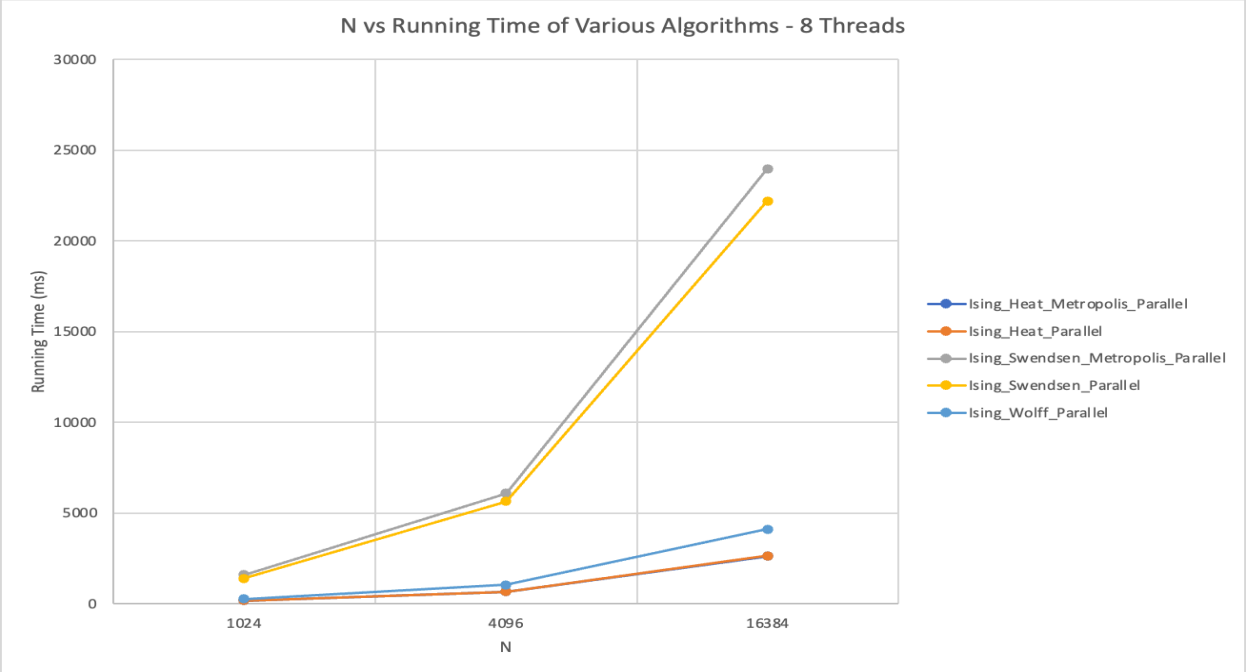

Figure 3: N vs Running Time for All Tested Algorithms - 8 Threads

Our analysis shows a general pattern where the speedup ratios, especially in algorithms using 4 and 8 threads, typically hovered near 1 or dropped to about 0.5. This indicates that doubling the

number of threads often did not halve the computation time, suggesting suboptimal parallel efficiency.

In terms of efficiency, which we defined as the ratio of speedup to the number of threads, we observed consistent outcomes where the efficiency matched the ratio of the serial runtime to the number of threads. These results imply that the performance gains from parallelization were limited and did not scale linearly with increased thread counts.

Algorithm	N	Speedup (1:4)	Speedup (1:8)	Efficiency (Speed: 4 Thread)	Efficiency (Speed: 8 Thread)
Ising_Heat_Metropolis_Parallel	1024	1.00606	1.00606	0.25152	0.12576
	4096	0.98201	0.98496	0.24550	0.12312
	16384	0.98010	0.97562	0.24502	0.12195
Ising_Heat_Parallel	1024	1.05357	1.07273	0.26339	0.13409
	4096	0.95626	0.95482	0.23906	0.11935
	16384	0.97522	0.96894	0.24380	0.12112
Ising_Swendsen_Metropolis_Parallel	1024	0.38520	0.28483	0.09630	0.03560
	4096	0.39630	0.30003	0.09908	0.03750
	16384	0.42780	0.31414	0.10695	0.03927
Ising_Swendsen_Parallel	1024	0.35493	0.27707	0.08873	0.03463
	4096	0.36051	0.27152	0.09013	0.03394
	16384	0.36377	0.27743	0.09094	0.03468
Ising_Wolff_Parallel	1024	0.99628	1.02290	0.24907	0.12786
	4096	1.01850	1.01258	0.25463	0.12657
	16384	0.99586	0.99756	0.24897	0.12470

Table 1: N vs Speedup and Efficiency for All Tested Algorithms

4.2 Discussion

Based on the results, we can see that the parallelization of the Swedsen-Wang algorithm was not achieved and often had the same or worse performance than the serial version. However, this is because the Swedsen-Wang algorithm primarily used serial-oriented loops. What this means is that the loops in the algorithm are designed in such a way that they have better performance running in serial rather than parallel. This can be seen in the results where the serial version of Swedsen-Wang often performed much better than its parallel counterpart. This does mean that it could be possible to parallelize both algorithms to have better performance than their serial versions, just not through the method we employed to parallelize them.

5 Conclusion

In conclusion, our endeavor to optimize Monte Carlo simulation algorithms through parallelization techniques has provided valuable insights into the intricacies of computational efficiency. While our focus primarily revolved around the Metropolis, Swendsen-Wang, and Wolff algorithms, our attempts at parallelization, particularly with the Swendsen-Wang algorithm, encountered challenges stemming from its intrinsic design.

References

Chib, Siddhartha, and Edward Greenberg. “Understanding the Metropolis-Hastings Algorithm.” *The American Statistician*, vol. 49, no. 4, 1995, pp. 327–35. *JSTOR*, https://doi.org/10.2307/2684568.

D. Frenkel and B. Smit (2002) Understanding Molecular Simulation. San Diego: Academic, 2nd ed.

U. Wolff (1989) Collective Monte Carlo updating for spin systems. Phys. Rev.Lett. 62(4), pp. 361–364

“Metropolis-Hastings Algorithm.” Wikipedia, Wikimedia Foundation, 18 Apr. 2024, en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm.

“Swendsen–Wang Algorithm.” Wikipedia, Wikimedia Foundation, 28 Apr. 2024, en.wikipedia.org/wiki/Swendsen%E2%80%93Wang_algorithm.

“Wolff Algorithm.” Wikipedia, Wikimedia Foundation, 30 Oct. 2022, en.wikipedia.org/wiki/Wolff_algorithm.