Gregory Maldonado

Parallelizing and Benchmarking Monte Carlo Integration

Monte Carlo integration is the process of selecting random uniformly distributed numbers between the lower and upper bounds of a definite integral with each [pseudo-]random number generated being input to the function. The approximated integral value of the function is the summated total of each function output divided by the total number of samples taken. The integration calculation was benchmarked on two machines, a local MacBook Air M2 with 8 cores (4 performant and 4 efficiency cores), and the Binghamton University OpenHPC servers. Both benchmarks computed 100,000,000 [pseudo-]random numbers and took the average of three runs, and increased the number of threads for each sample taken.

Results on the Binghamton University OpenHPC Servers

The first benchmark taken was speed-up on the Binghamton University OpenHPC server, requesting 8 cores with 500 MB of memory through a SLURM job. Speed-up is the proportionality of the compute time of a single thread and the compute time using *n* threads. The speed-up on the OpenHPC servers was approximately proportional to the number of the threads used within the Monte Carlo simulation as shown in figure 1. Monte Carlo integration is an ideal use case for multithreading; if the number of threads are doubled, then the compute time is halved.

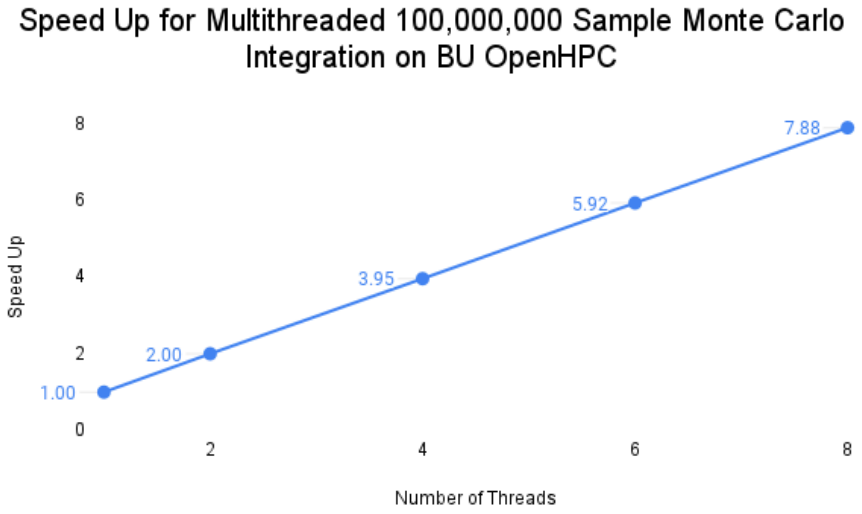

Figure 1. Speed Up for Multithreaded 100,000,000 Sample Monte Carlo Integration on Binghamton University OpenHPC Servers

Similarly, the efficiency of the Monte Carlo integration on the Binghamton University OpenHPC servers is the total compute time for *n* threads over the number of threads used for the simulation - if a program is efficient, then the efficiency should remain constant. This is a representation of how efficient the program was with each thread. The Monte Carlo integration simulation effectively used each thread and recorded

an approximate efficiency of 1.0 as shown in figure 2. Further optimization and compiler flags could be made to get the efficiency closer to 1.0, but were not explored during this round of sampling.

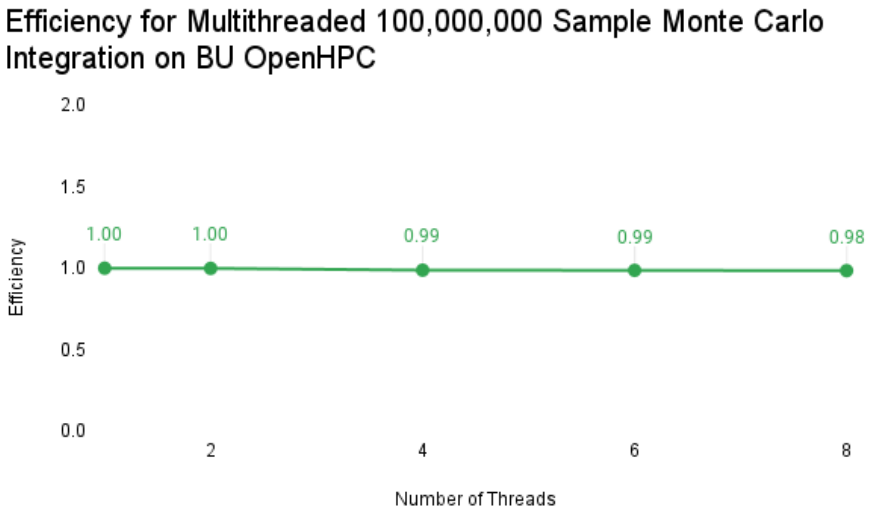

Figure 2. Efficiency for Multithreaded 100,000,000 Sample Monte Carlo Integration on Binghamton University OpenHPC Servers

Results on a MacBook Air M2, 4 Performant, 4 Efficiency Core Machine

The same program was compiled (albeit for ARM architecture) with the same metrics of 100,000,000 samples running between one and eight threads. On MacBook M2, an approximately linear relationship can be observed between speed-up and number of threads, as shown in Figure 3, closely resembling results found on the Binghamton University OpenHPC servers.

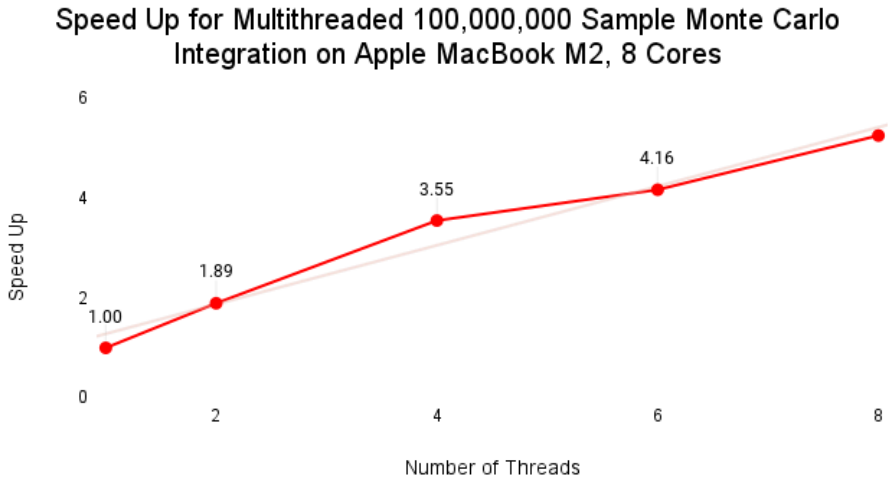

Figure 3. Speed for Multithreaded 100,000,000 Sample Monte Carlo Integration on Apple MacBook M2, 4 Performant Cores and 4 Efficiency Cores

The more interesting benchmark being the efficiency of the Monte Carlo integration on the MacBook M2 machine; the relationship between the number of threads and the efficiency can be observed to be negatively logarithmic as shown in Figure 4.

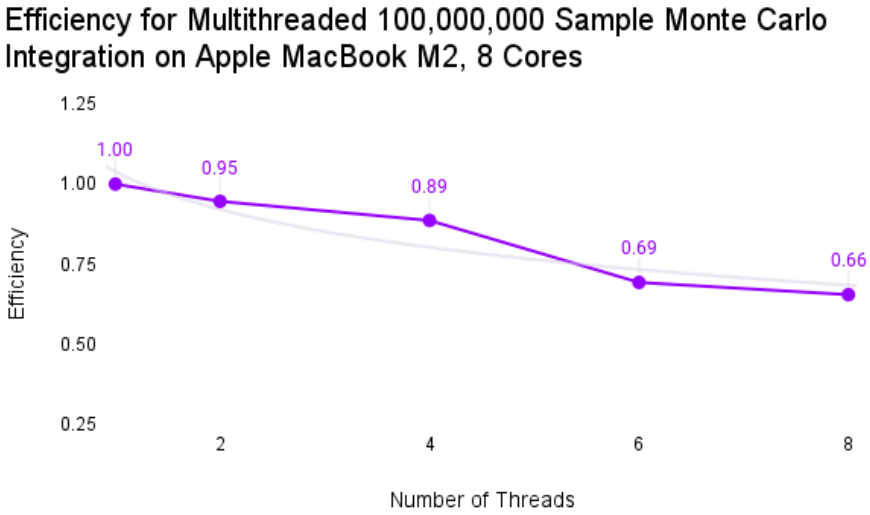

Figure 4. Efficiency for Multithreaded 100,000,000 Sample Monte Carlo Integration on Apple MacBook M2, 4 Performant Cores and 4 Efficiency Cores

While the program was executed on both the Apple MacBook M2 and the Binghamton University OpenHPC server and showed similar speed-up results, the efficiency was drastically different. This discrepancy is most likely due to the M2's four efficiency cores. The Monte Carlo simulation does not hit the threshold for computation power to activate all cores on the machines causing only four hardware threads to be spawned and the rest of the requested threads to act as virtual threads. This can be further explained by the drastic dip in efficiency after four threads are requested.