Justin Bahr
NUID: 001592707

High Performance Computing
Homework 2 - Question 1

In addition to the Monte Carlo estimation of pi, I also wrote a program to calculate pi using the Leibniz formula. I ran my programs on the COE Vector Linux platform (Intel ® Xeon ® CPU E5-2698 v4 @ 2.20GHz, 20 cores per socket, 2 sockets, with 791957492 KB of memory, and Linux 4.18.0).

a. For the Pthread programs, I evaluated speedup using a test case of 10 million darts (or terms for the Leibniz method), checking the runtime with 1, 2, 4, 8, 16, 32, 64, and 128 threads. The table below shows the approximate optimal number of threads to use for this problem size and the speedup obtained compared to the single-thread base case:

Program	Optimal Thread Number	Speedup Obtained
Monte Carlo	128	28.3833
Leibniz	32	4.96889

Figure 1. Pthread Program Speedups

b. For the OpenMP programs, I evaluated speedup using a test case of 10 million darts (or terms for the Leibniz method), checking the runtime with 1, 2, 4, 8, 16, 32, 64, and 128 threads. The table below shows the approximate optimal number of threads to use for this problem size and the speedup obtained compared to the single-thread base case:

Program	Optimal Thread Number	Speedup Obtained
Monte Carlo	64	27.7706
Leibniz	32	12.8555

Figure 2. OpenMP Program Speedups

I suspect the reason that the Monte Carlo estimation had a much larger speedup was because the dart throws were executed using random ints and a comparison function. The integers take up less space in memory than doubles and the comparisons can be done concurrently by multiple threads. The Leibniz method requires each thread to update a double variable as a local sum.

c. Parts a and b address strong scaling for each program for a fixed problem size of 10 million. Pthreads provided a slightly larger speedup for the Monte Carlo estimation, and OpenMP provided a significantly larger speedup for the Leibniz method. In general, the Leibniz program began to slow down after adding more than 32 threads, but the Monte Carlo program did not appear to slow down when more than 32 threads were used. Plots relating number of threads used to runtime are shown below:

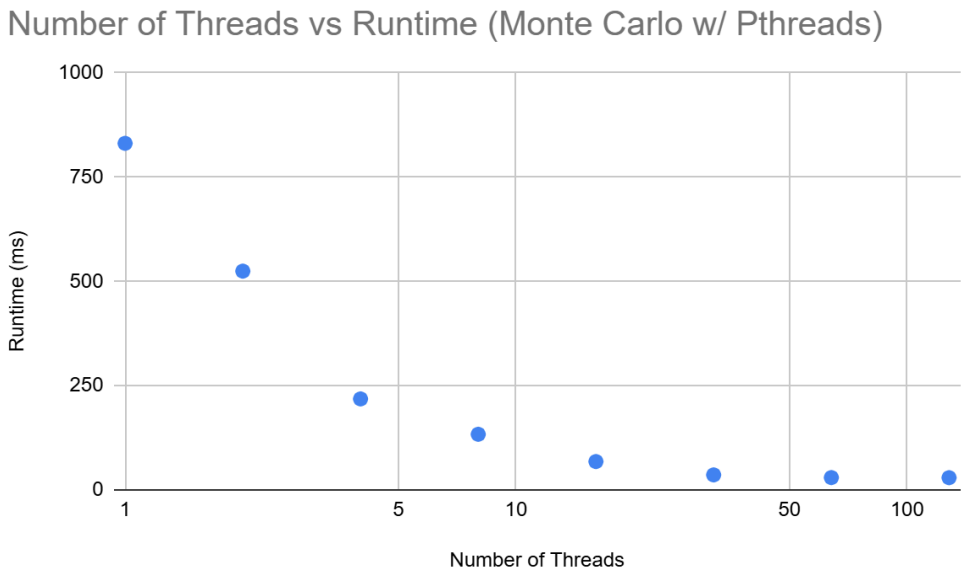

Figure 3. Number of Threads vs Runtime (Monte Carlo w/ Pthreads)

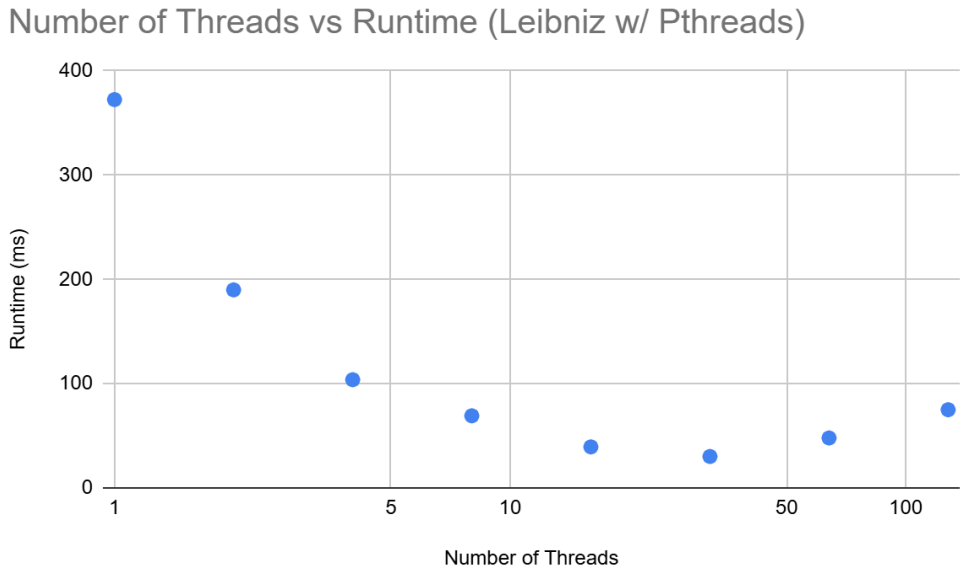

Figure 4. Number of Threads vs Runtime (Leibniz w/ Pthreads)

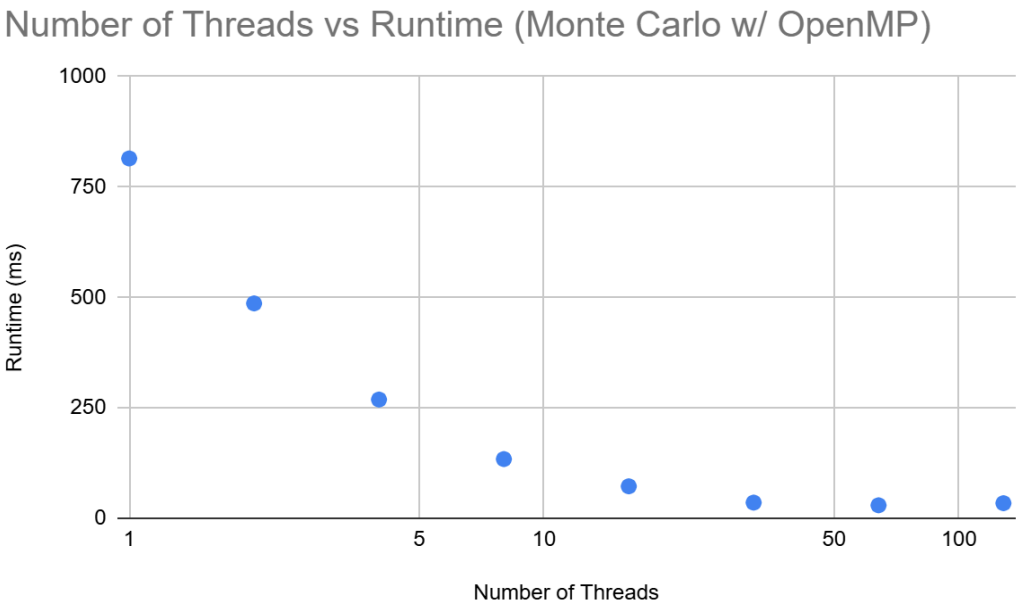

Figure 5. Number of Threads vs Runtime (Monte Carlo w/ OpenMP)

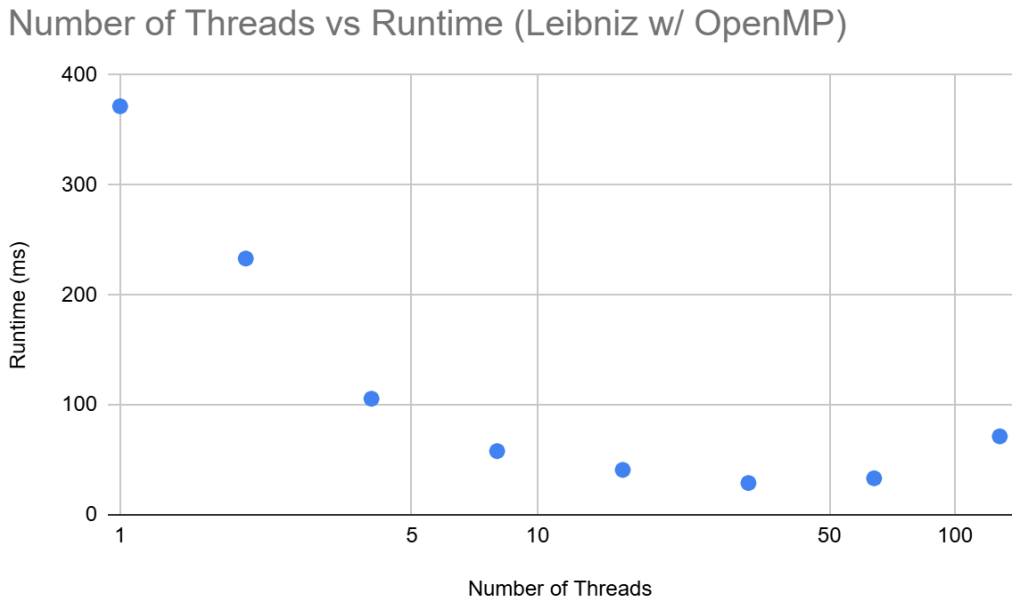

Figure 6. Number of Threads vs Runtime (Leibniz w/ OpenMP)

To evaluate weak scaling, I tested each program with the same numbers of threads as before, but I always used 100,000 darts/terms per thread. Then I calculated the run's throughput in dart/terms per microsecond. The table below shows the approximate maximum throughput and the respective number of threads used to achieve this:

Program	Threads	Throughput (darts/terms per microseconds)
Monte Carlo (Pthread)	128	352.7924459
Leibniz (Pthread)	128	2375.534078
Monte Carlo (OpenMP)	64	324.8924581
Leibniz (OpenMP)	64	2304.362338

Figure 7. Maximum Observed Throughput

Weak scaling was very effective up to a point for both programs. There was an approximately linear increase in throughput with respect to the number of threads while 32 or less. After the number of threads was increased to 64, the increase in throughput began to be less noticeable or even decrease in the case of OpenMP. This is likely because the node I was testing on had 40 physical cores, so increasing the number of threads beyond this number does not utilize any additional hardware. The throughput plots I used to evaluate weak scaling for each program are shown below:

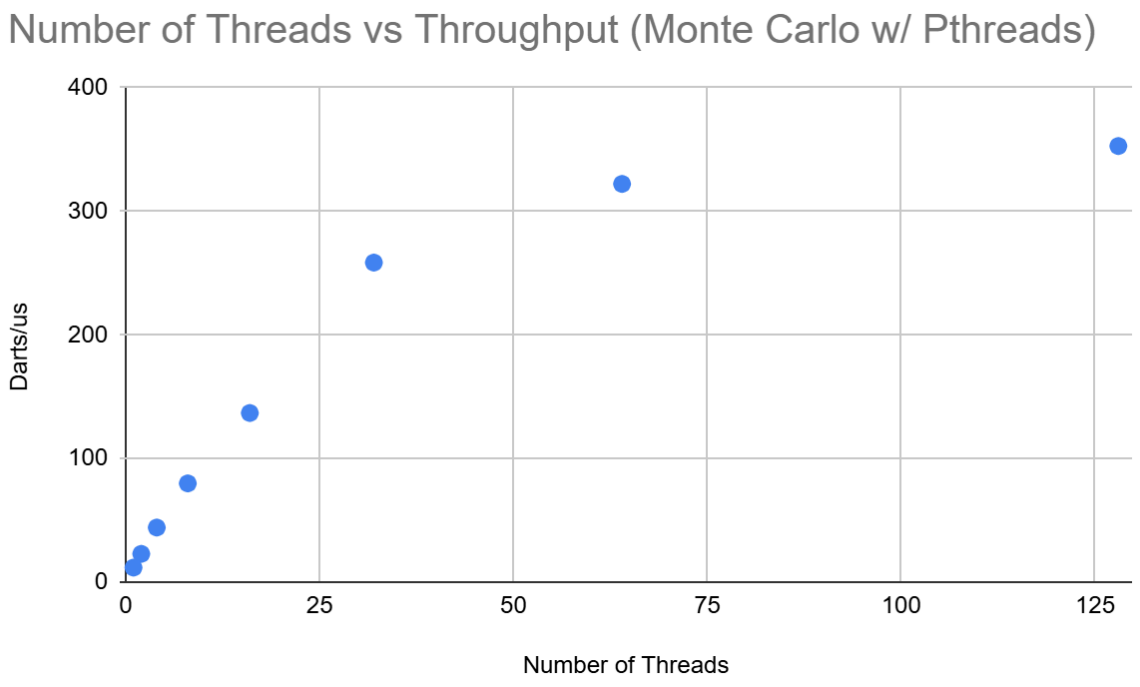

Figure 8. Number of Threads vs Throughput (Monte Carlo w/ Pthreads)

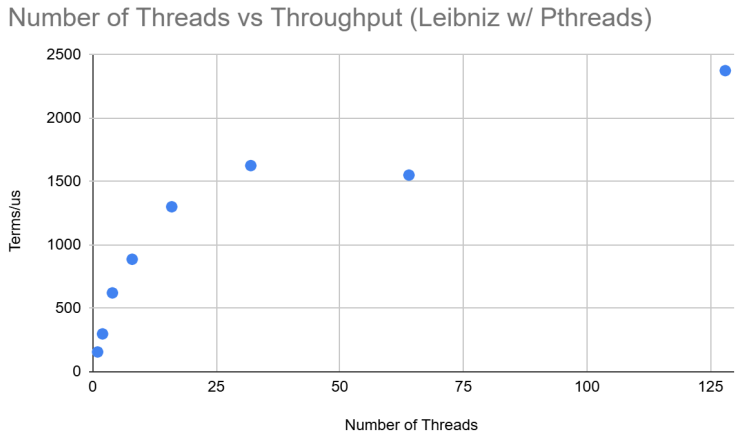

Figure 9. Number of Threads vs Throughput (Leibniz w/ Pthreads)

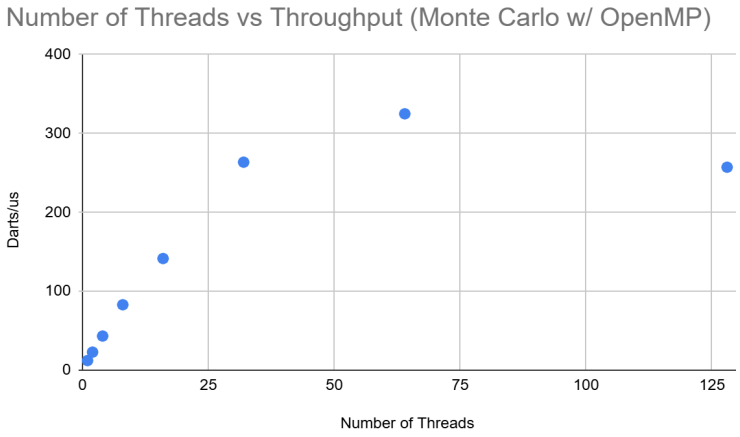

Figure 10. Number of Threads vs Throughput (Monte Carlo w/ OpenMP)

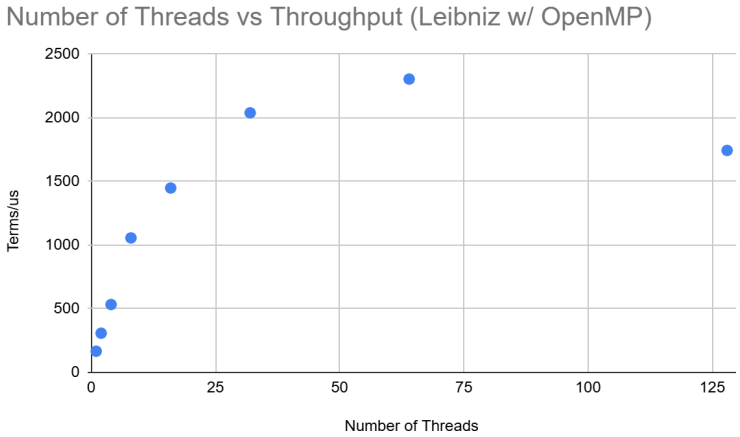

Figure 11. Number of Threads vs Throughput (Leibniz w/ OpenMP)

d. The Leibniz method was far superior in terms of accuracy. It had a lower absolute error for any amount of terms compared to the Monte Carlo estimation. Additionally, the Leibniz method converged to 3.14159, while the Monte Carlo estimation seemed to converge to about 3.137. Since the Leibniz method is based on an exact formula, adding more terms to the sum will always increase the accuracy. However, since the Monte Carlo estimation is based on a randomly generated dataset and a quantized dart board, it is not guaranteed to increase in accuracy when more darts are thrown. Here is a plot of absolute error with respect to number of darts/terms for both methods:

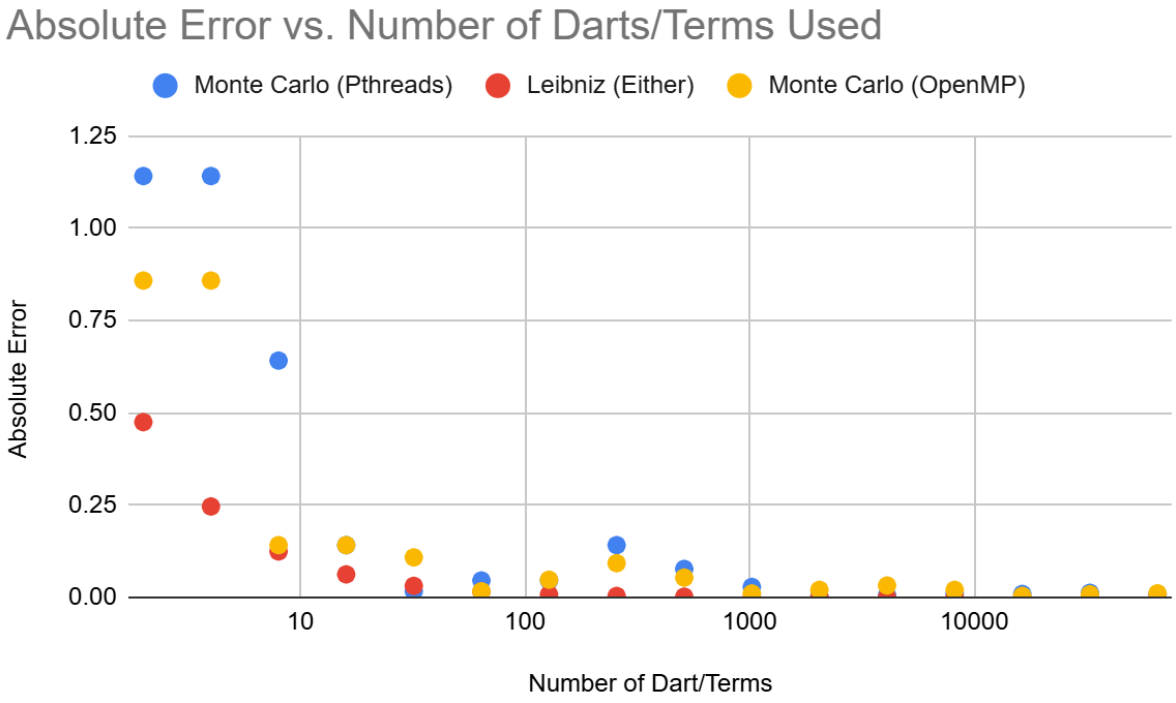

Figure 12. Absolute Error of Each Method