Manuel Berrueta
Ryan Neisess

Programming Project #4: Pi Estimator (using OpenMP multithreading)

This Pi Estimator assignment simulated a dart thrown n times into a unit square and by computing the fraction of the times that the dart fell into the unit circle enclosed within the unit square and multiplying it by 4, we got an estimate of Pi. We expected that for the larger input n, we would get a higher level of accuracy on our Pi estimator code. We also expected that by using a higher amount of threads we would get a scalable speed up, that is that the amount of time it took to process the estimate would be sufficiently faster by taking advantage of multiple threads, which would also show that our code scales properly.

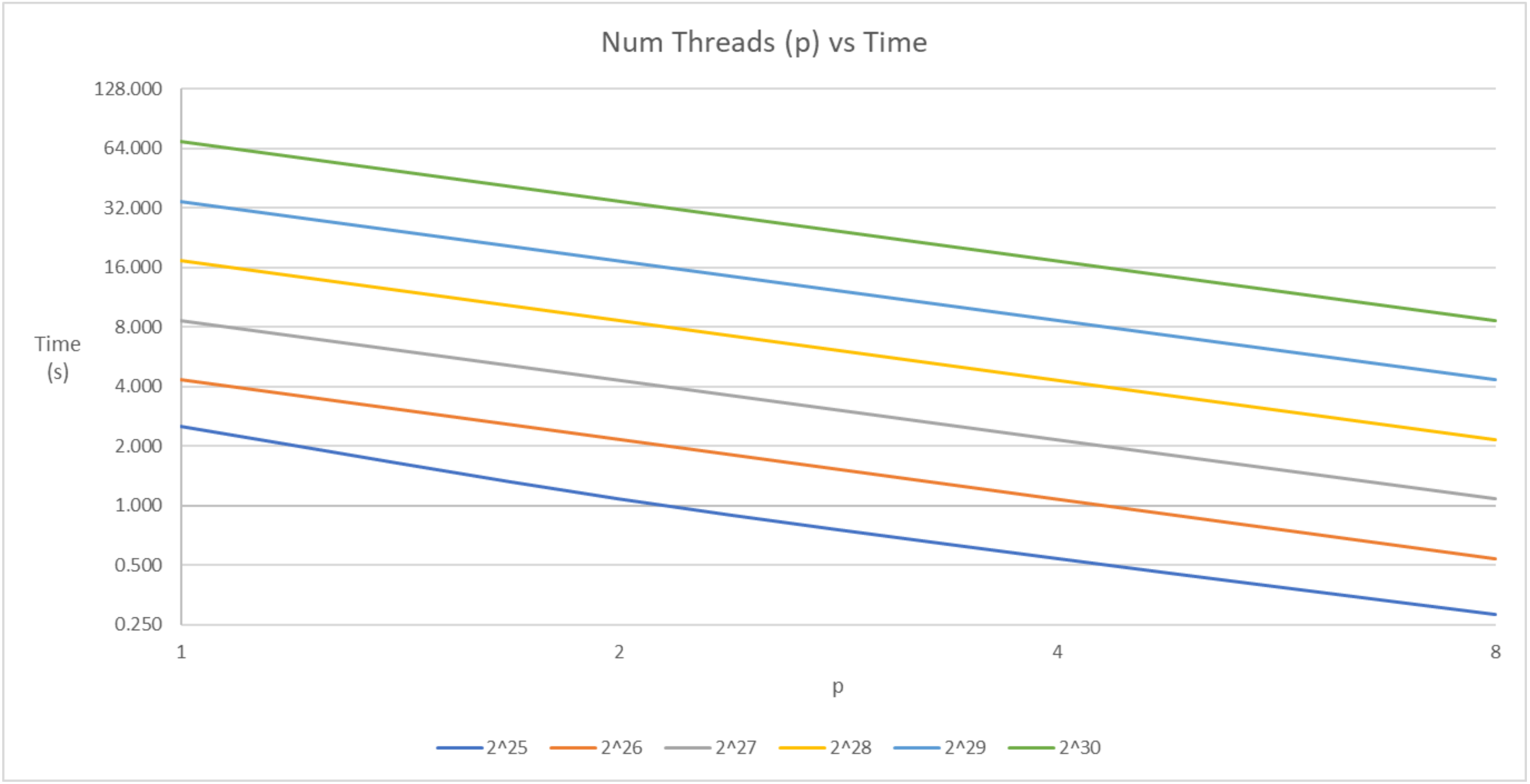
We faced some challenges with understanding the way OpenMP worked and we had to modify the code a few times to get it to work properly. At times it would not scale right, the times didn't look right, and we had to do some debugging and get a better understanding of what was happening. At one point we realized that one of the issues we were having was being introduced by the job shell script we had, and after tweaking that we got it to work properly. There was a time where the higher the input n would give us worst accuracy, then half that amount which was not what we were expecting it all and it wasn't after working with the code that we were able to move past that blocking point.

Moving along to our test findings. For experiment 1, I have to say that this was the first time that we were able to get consistent times with the cluster, every other time we gotten mixed times and it wasn't as easy to see the pattern that we should get would be as accurate as the one shown in the table below:

	p			
n	1	2	4	8
33,554,432	2.507	1.080	0.540	0.282
67,108,864	4.320	2.161	1.080	0.541
134,217,728	8.641	4.321	2.161	1.089
268,435,456	17.275	8.643	4.322	2.171
536,870,912	34.563	17.285	8.643	4.328
1,073,741,824	69.121	34.571	17.285	8.649

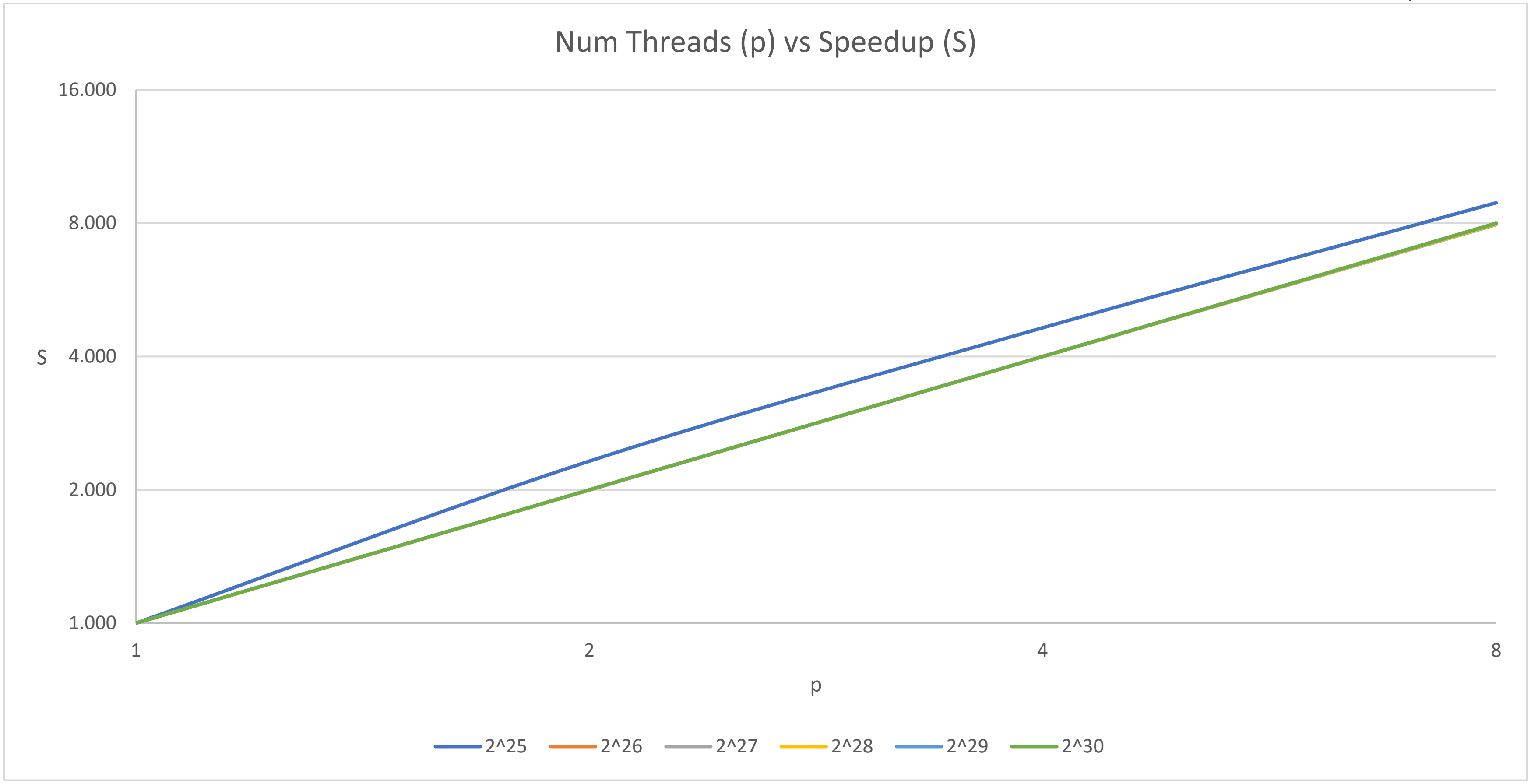
Table 1. Raw times for calculation of pi for increasing number of threads for fixed input sizes.

Manuel Berrueta
Ryan Neisess



	n					
p	33554432	67108864	134217728	268435456	536870912	1073741824
1	1.000	1.000	1.000	1.000	1.000	1.000
2	2.321	1.999	2.000	1.999	2.000	1.999
4	4.643	4.000	3.999	3.997	3.999	3.999
8	8.890	7.985	7.935	7.957	7.986	7.992

Table 2. Speedup values for increasing number of threads for fixed input sizes.



As you can see from the table the code from fixed values of n and by doubling the threads, the speedup we achieved was even better than what we expected due to the number we have seen before on the cluster, this number were achieving about a 2x speed up each time the threads were doubled. It is also a worthy note that by doubling n each time we are able to see that the code scales properly and the number match nicely diagonally, which was very impressive to us and a result/behavior we were not able to achieve until today. Needless to say, this is what good scalable good leveraging parallel work should show.

For experiment 2, on our second below table we keep n/p fixed, that is the ratio of n/p and then we doubled the n/p ratio each time as per experiment 2. We were expecting accuracy to increase substantially, but our results show otherwise. The most consistent accuracy was that of 5 decimal places with our largest number/thread combination of 1073741824 / 32. The time also started double after 8 threads consistently up to our last value of 32 which was an interesting find but not unexpected.

n / p	Time	Estimated Value	Num Decimal Places
33554432 / 1	2.16	3.141729474	3
67108864 / 2	2.161	3.141760826	3
134217728 / 4	2.161	3.141499013	3
268435456 / 8	2.173	3.141555622	4
536870912 / 16	4.345	3.141560383	4
1073741824 / 32	8.656	3.141599115	5

Table 3. Accuracy and precision of calculated values of pi for increasing n at fixed n / p ratio.

The project itself prove to be very useful to show the power of parallel code and scalability and since we got good clean and consistent numbers, it visually and consistently follows from the theory shown in class from the beginning until now what you are to expect and the things you are to take into account when you use parallel code.