Cpt S 411 Assignment Cover Sheet

Assignment #4

Team Members:

Lorphanpaibul, Allison

Nate Jensvold

I[[1]](#footnote-0) certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment. I also certify that I have not referred to online solutions that may be available on the web or sought the help of other students outside the class, in preparing my solution. I attest that the solution is my own and if evidence is found to the contrary, I understand that I will be subject to the academic dishonesty policy as outlined in the course syllabus.

Please print your names.

Allison Lorphanapaibul

Nate Jensvold

Today’s Date:

November 17, 2020

1. **Introduction**

For this assignment, we used openmp to estimate the value of pi. For our performance measures, we used the total runtime and the precision of the pi estimate to test our program. Both speedup and precision tests were done on a single core of the cluster. Since each core only has eight threads, we only tested up to eight threads.

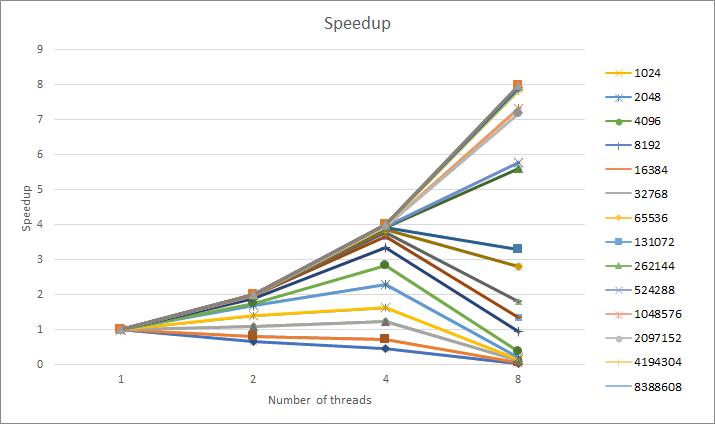
1. **Speedup Analysis**

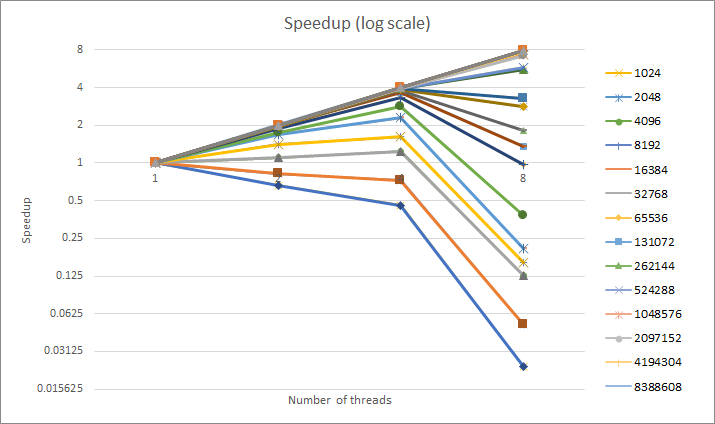
Below is our total runtime data. From the runtime data, we calculate the speedup.

We used the open mp time function to calculate the times, and the times are in seconds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Runtime | | | | | |
| Number of iterations | Number of threads | | | | |
|  | 1 | 2 | 4 | 8 |
| 1024 | 0.000122 | 0.000183 | 0.000265 | 0.005117 |
| 2048 | 0.000178 | 0.000217 | 0.000245 | 0.003442 |
| 4096 | 0.000317 | 0.000287 | 0.000256 | 0.002511 |
| 8192 | 0.000578 | 0.000411 | 0.000356 | 0.003579 |
| 16384 | 0.001114 | 0.000656 | 0.000485 | 0.005327 |
| 32768 | 0.002149 | 0.001229 | 0.000755 | 0.005598 |
| 65536 | 0.004265 | 0.00225 | 0.001273 | 0.004423 |
| 131072 | 0.008575 | 0.004338 | 0.002338 | 0.006297 |
| 262144 | 0.016902 | 0.008576 | 0.004455 | 0.009277 |
| 524288 | 0.033754 | 0.017101 | 0.008705 | 0.011977 |
| 1048576 | 0.067336 | 0.034062 | 0.017152 | 0.02038 |
| 2097152 | 0.134581 | 0.06799 | 0.034283 | 0.024016 |
| 4194304 | 0.269049 | 0.135563 | 0.068035 | 0.046436 |
| 8388608 | 0.537849 | 0.270863 | 0.135828 | 0.073333 |
| 16777216 | 1.07542 | 0.538522 | 0.271069 | 0.149994 |
| 33554432 | 2.150893 | 1.07663 | 0.53869 | 0.274387 |
| 67108864 | 4.334415 | 2.152927 | 1.07784 | 0.551137 |
| 134217728 | 8.603181 | 4.305009 | 2.152647 | 1.094048 |
| 268435456 | 17.205461 | 8.573884 | 4.305472 | 2.169212 |
| 536870912 | 34.416063 | 17.146378 | 8.573849 | 4.310866 |
| 1073741824 | 68.828944 | 34.296684 | 17.146827 | 8.627691 |

After calculating the speedups, we were able to create the following speedup graphs. The first graph shows the speedup for each number of iterations, and the seconds shows the same thing with a log scale.



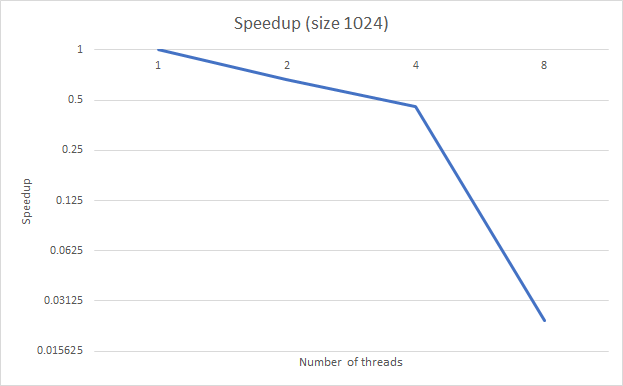


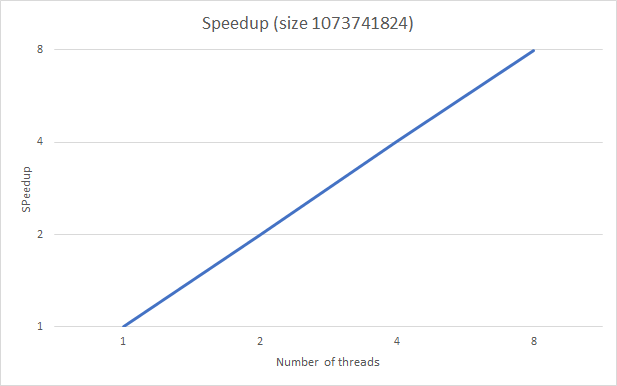
From these graphs, we observed that the speedup between threads was more noticeable at higher input numbers. At lower inputs, the speedup was negligible or even worse with increased threads. Overall, the data was as expected. As threads increased by powers of two, the time would be half.

Below are the isolated lines of the lowest and highest input sizes, for easier viewing.

You can see here how the speedup goes down as the number of threads increases for the lower input, and the opposite happens for the higher input.

Our speedups for the highest input were quite good, resulting in a pretty much linear/ideal speedup curve.



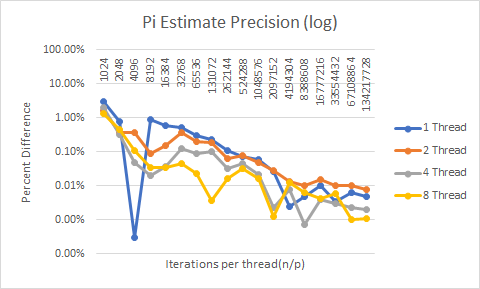
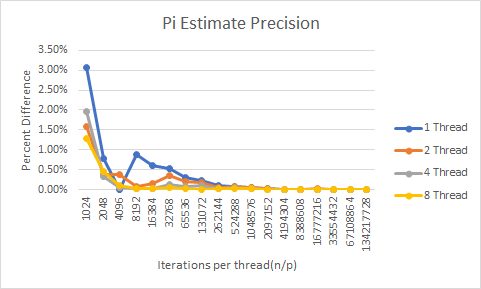


1. **Precision Testing**

For precision testing, we kept track of the value of pi for each different iteration of the program. To calculate how well the program estimated pi, we calculated the percent difference of the estimated pi and the actual value of pi.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Estimation of pi | | | | | |
| Number of iterations | Number of threads | | | | |
|  | 1 | 2 | 4 | 8 |
| 1024 | 3.238281 | 3.285156 | 3.214844 | 3.203125 |
| 2048 | 3.166016 | 3.191406 | 3.195312 | 3.191406 |
| 4096 | 3.141602 | 3.15332 | 3.203125 | 3.209961 |
| 8192 | 3.11377 | 3.15332 | 3.151855 | 3.182129 |
| 16384 | 3.122559 | 3.144287 | 3.143066 | 3.155518 |
| 32768 | 3.124878 | 3.136963 | 3.140991 | 3.14502 |
| 65536 | 3.131958 | 3.130249 | 3.142761 | 3.142639 |
| 131072 | 3.134552 | 3.135468 | 3.137726 | 3.140533 |
| 262144 | 3.138123 | 3.135757 | 3.138885 | 3.140167 |
| 524288 | 3.139351 | 3.139648 | 3.138336 | 3.140892 |
| 1048576 | 3.139759 | 3.139225 | 3.140568 | 3.141708 |
| 2097152 | 3.14076 | 3.140078 | 3.140207 | 3.141069 |
| 4194304 | 3.141515 | 3.140715 | 3.140936 | 3.140568 |
| 8388608 | 3.141744 | 3.141185 | 3.141519 | 3.141075 |
| 16777216 | 3.14192 | 3.141273 | 3.141841 | 3.141554 |
| 33554432 | 3.141488 | 3.14111 | 3.14157 | 3.141974 |
| 67108864 | 3.141402 | 3.141286 | 3.141466 | 3.141794 |
| 134217728 | 3.141443 | 3.14191 | 3.141498 | 3.141463 |
| 268435456 | 3.141792 | 3.141833 | 3.141663 | 3.141412 |
| 536870912 | 3.141606 | 3.141564 | 3.141654 | 3.141624 |
| 1073741824 | 3.141641 | 3.141612 | 3.141601 | 3.141558 |

After calculating the percent differences, we graphed the results to show how the percent different changes as n increases. We chose data to graph where n/p is constant, as highlighted in the table above.

****

Viewing the graphs above, it can be seen that the percent difference between our program’s estimate and the actual value of pi went down as the number of iterations (n) increased. These results were expected.

**Conclusion:**

Looking at our data a few obvious trends begin to emerge. The first being that the number of threads used has no meaningful impact on the precision of PI. The precision of PI on 1 thread started at a 3.0777% difference at n/p=1024 and went all the way down to 0.0048% when n/p=134217728. Similarly when looking at the precision of PI on 8 threads the percent difference started at 1.2903% at n/=1024 and went down to 0.0011% when n/p=134217728. Another trend that stood out was the speedup found at high iterations. On 1024 iterations the 8 thread performance was 0.02 the speed of the 1 thread performance, however on 1073741824 iterations that speedup was almost completely linear and shot up to 7.97 faster than the 1 thread performance.

The trends that we encounter as we look at our data are in line with what we expected when we started this assignment. Speedup should be almost linear at a large number of iterations because the overhead of initializing and setting up the environment for each thread will be a smaller portion of the total runtime of the program. Additionally it makes sense that the speedup on small iterations would be abysmal because the aforementioned setup time across each thread is larger than the amount of time it takes to iterate a small number of times. Looking at the PI precision data the trend pointed out above is what is to be expected when n/p is a fixed number. PI precision data is tied to the total number of iterations across all threads regardless of how the iterations are split up. As total iterations increase the precision of PI will continue to go up as more darts are thrown into the unit circle/square.

While we were working on this project there were several major issues that we ran into that had to be addressed. The first of those issues was when we first started running the program on the cluster we were calling the program with mpirun. When the program was getting called with mpirun we noticed that there was no speedup for any number of iterations no matter how large. Another issue that we ran into was when we were generating our random number we started by using the rand() function instead of srand48\_r(). The reason that this was an issue was because the rand() function is not thread safe when it generates numbers causing a massive hit in performance when it is used across multiple threads. When we used rand() very little speedup was observed as the number of iterations went up until we switched out the function with the threadsafe srand48\_r(). The final issue that we observed was the value of PI that we were generating when we used variables inside the threaded code. When we were generating random numbers and storing them with additional variables we noticed that no matter how many iterations were performed we always got an estimated value of PI =~2.8. We never figured out the exact reason behind this phenomenon but we were able to solve the issue when we refactored out code to use as few variables as possible inside of the threaded code.

1. If you worked as a team, then the word “I” includes yourself and your team members. [↑](#footnote-ref-0)