The higher the process count, the longer each program takes to “set up”. By the time you get to 12 processes, it is a noticeable length of time that you are waiting. I don’t really notice much by way of execution time, but this is a fairly small data set. I would love to see some benchmarking tools, but I have only ever done that in python and wouldn’t know where to start with this program since I have consistently seen that the various processes start before the first user prompt is executed.

~~My trial runs were not very accurate because for some reason, every random number generated is the exact same number. From what I know about pseudo-random generated numbers and C, I cannot see why this is happening. My only thought is that this is being seeded at close enough to the same time in each process that the sequence being called for numbers is the same sequence for each process. I do not know how to get around that issue.~~

While I showed Blake how to generate the random numbers, he corrected my issue in seeding the random generator to correct my estimations. Gratefully, by the happenstance of general statistics, the more throws and processes you use, the closer the number should get to pi. ~~Due to the errors in my random generated numbers, this is not the case. The number is actually closer the more throws and the FEWER processes I used only because it allowed for more random numbers to be generated instead of reused by the other processes.~~

After today’s lecture, I will be editing this later to see if I can implement some of the benchmarking and MPI\_Barrier to clean up and synchronize the processes for further organization and see if that changes the time and estimate values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Throws** | **2 Processes** | **4 Processes** | **8 Processes** | **12 Processes** |
| 24 | 3.500000 | 3.000000 | 2.833333 | 3.000000 |
| 500 | 3.168000 | 3.112000 | 3.208000 | 3.104000 |
| 5000 | 3.161600 | 3.125600 | 3.180000 | 3.152800 |
| 100000 | 3.138080 | 3.151720 | 3.135160 | 3.131960 |
| 250000 | 3.141232 | 3.143792 | 3.139424 | 3.142352 |
| 1000000 | 3.140704 | 3.140292 | 3.143336 | 3.140936 |