CS426 Project 2

Barış Can 21501886

1 Implementation and Design Choices

For the serial implementation, I generated random numbers between 0 and 1, normalized them by subtracting 0.5 since I want them to be at the center of the circle and took the square of them. In my implementation, arr[0] corresponds to x and arr[1] corresponds to y, so I added and took square-root since I wanted to calculate the distance from the center. If the distance is bigger than the r, namely 0.5, then it is not inside the circle and the range must be between 0 and $0.5\sqrt{2}$ since it is the diagonal of the square. Then, I calculated the hit points and probability. My probability was generally between 0.77 and 0.80 and gave better results when I increase the number of experiments. At the last experiment, results was generally between 0.785 and 0.786 which is very close to $\pi/4$ and 0.78539. Time results will be evaluated in the discussion part.

In the parallel part, I think there were several misunderstandings. I changed my implementation many times according to mails come from TA's and version two of the project. Therefore, this part of the report will be a bit complicated since I want to convey my thoughts and different implementations of the project. First of all, I created functions Map, Fold and Filter, and I made them parallel with MPI. Then I read the second part of the assignment and saw that we also need to divide the number of experiments somehow. Since I used MPI in the first part, I did not understand how we can use MPI in the second part as well. I thought four ways. Firstly, the sentence could be wrong and we should not divide the number of experiments since we already divide them in the functions. Secondly, we could divide the number of experiments but not divide the functions (although this is the most effective strategy for me, in the assignment, it explicitly says we must use communication schemes in the functions). These two solutions would cause data parallelism. Thirdly, we could divide both experiments and functions. For example, we have 16 processes and 100 experiments, first 4 of the processes could handle 25 of the experiments, and inside the functions, we could divide that 4 processes which would cause both data and task parallelism. However, inputs of the functions did not include any input such as the number of processes given, so I did not use this method. Lastly, we could divide which processes will handle which functions and this would cause task parallelism. The difference between data and task parallelism is, in the data parallelism, you divide the data and processes work on the same function, in the task

parallelism, you divide the functions and processes work on the same data considerings the data is independent. What I first did was something between the first and second thought. I understood that we should also divide the work caused by generating a random number since it is similar to the dividing number of experiments. Since Map is the first method that I used in my implementation, I passed an empty array to Map functions, and Map generated random numbers for me via parallelism. However, after the clarification on the assignment, it is said that we should not change the functionalities of functions. Therefore, I moved the generating random numbers part to the main and Map function only performed what it should perform. Apart from these, I used Map function to take the square of the random numbers, Filter to check if the distance is smaller than 0.5 and Fold to calculate the total hit points. I did all the other parts as I did in the serial implementation. Three function that I used inside these functions were square, add and isBigger. For the helper file, I used very similar things as the first project. I calculated the number of indexes to send to each process, send them via Send function since I thought this is the simplest and safer and faster way, and receive them by using Recv function. I calculated the functions that inside the functions. Lastly, I send the arrays that calculated in the processes to master, combine them and return them to the main. Map and Filter functions were very similar except the work I did in the processes. The whole architecture must be identical between them. I defined enum false and true to create bool type and used that inside the Filter function. Fold function was slightly different since every process returns one float number. I gather them in Master and calculate the total of the sums that comes from processes as I did in the minimize part of the first project. For the fold function, the only thing that was complicated for me was passing the initial value variable. I tried to pass arr[0] in the main method, but since it connected to very different variables somewhere else, I could not pass it, use it in the Send function. I really could not understand this part, I was able to print it in the console, but I could not use Send function. I send the array, indexes to processes, but I could not access processes. It gave segmentation faults again and again and I understood the cause after I wrote the function two times.

As for the address spaces in the questions, in MPI, each process has its own private address space. They are OS-level processes. However, processes can explicitly allocate shared memory regions. Moreover, all function pointers are in the processes' address space. Therefore, they are unique to one process and cannot be shared between processes if there is a truly distributed environment. For the MPI between different computers, as far as I understand from the searches I did on the Internet, MPI library distinguishes different computers from their MAC addresses and distribute the ranks according to MAC addresses and hostnames in OS. Therefore, MPI handles distinguishing different computers, giving the processes ranks and treat them as one computer.

2 Discussion

I think my parallel implementation was very inefficient but I could not find another way to implement what says in the assignment. I think the most efficient way would be dividing the number of processes between processes and implementing the methods serially. This would be the parallel version of our serial implementation, we could have just divide the number of experiments and do the same work in the processes. However, since we are doing parallelism in functions, there were a lot of array passing between functions and there was three different parallel implementation in the code whereas, in the serial version, all the work handled in several for loops. Another efficient implementation would be the third thought as I mentioned above that we divide both number of experiments and functions if we have many available processes. Considering the implementation I did on this assignment, as it can be seen from Figure 1, serial implementation was a bit faster than 4-process parallel implementation. 2-process parallel implementation was slightly faster than serial version but I did not want to compare it with the serial version since I thought comparing it with the 4-process would be better.

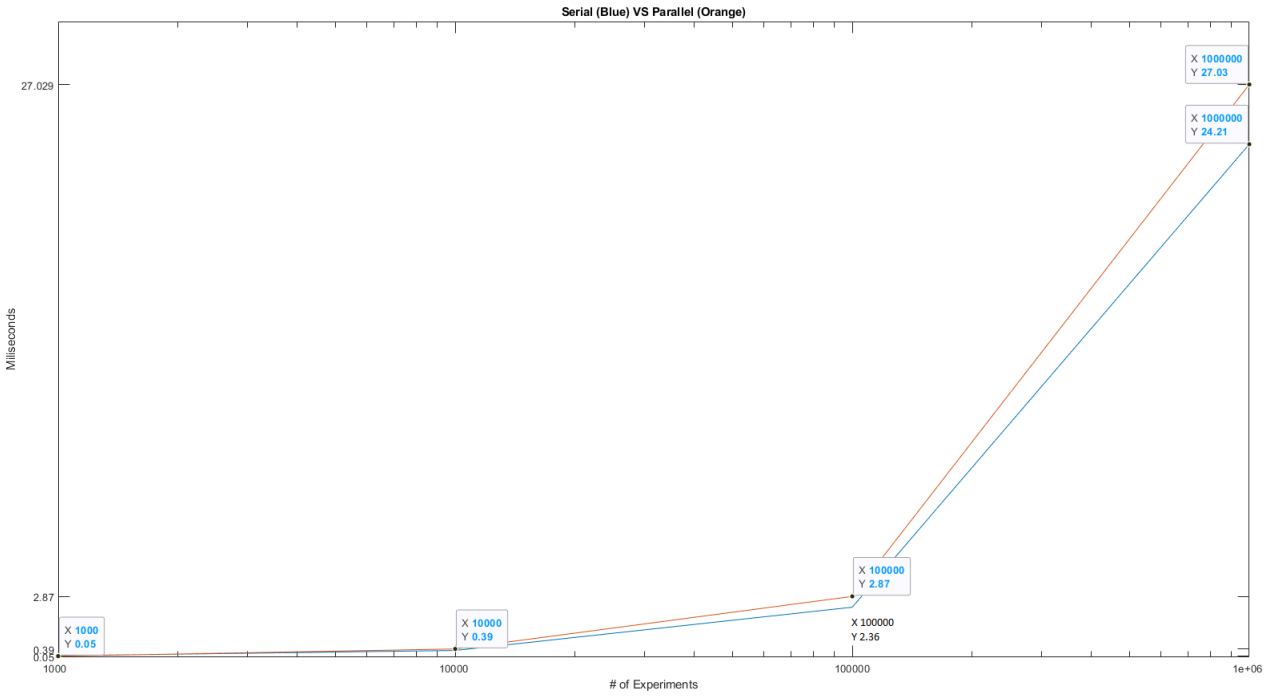

Figure 1: Comparison of the Serial and Parallel Implementation (4 Processes)

As it can be from Figure 1, orange represents parallel run and blue represents serial run. They were almost identical for the first two test and serial implementation was slightly faster in the last two tests.

I performed many experiments for the 9 process implementation and only put one of the average experiment in the figure, but for the first three tests of the experiment, I got very different results such as in one of them, 10000 was performed in 7 milliseconds and on the other, it was handled in 50 milliseconds. Sometimes the test has 100000 experiments ran faster than the test that has 1000 experiments since it took 48 milliseconds to complete. However, the last test that has 1000000 experiments was always higher than 100 milliseconds. I think the problem with the first three tests was the number of array passes bottleneck the parallelism so they were almost always between 30 and 50 milliseconds. In the main method, I could not control the parallelism, so every process created an array with the desired number of experiments, every process subtract 0.5 desired number of experiments times. The only part that I used parallelism was inside the functions in which there are three of them. I also tried 16-process but the elapsed times were so high that I did not want to put it on the figure since the test that has 1000000 experiments completed in 1 second. The overall average test results I mentioned can be seen in Figure 2 below.

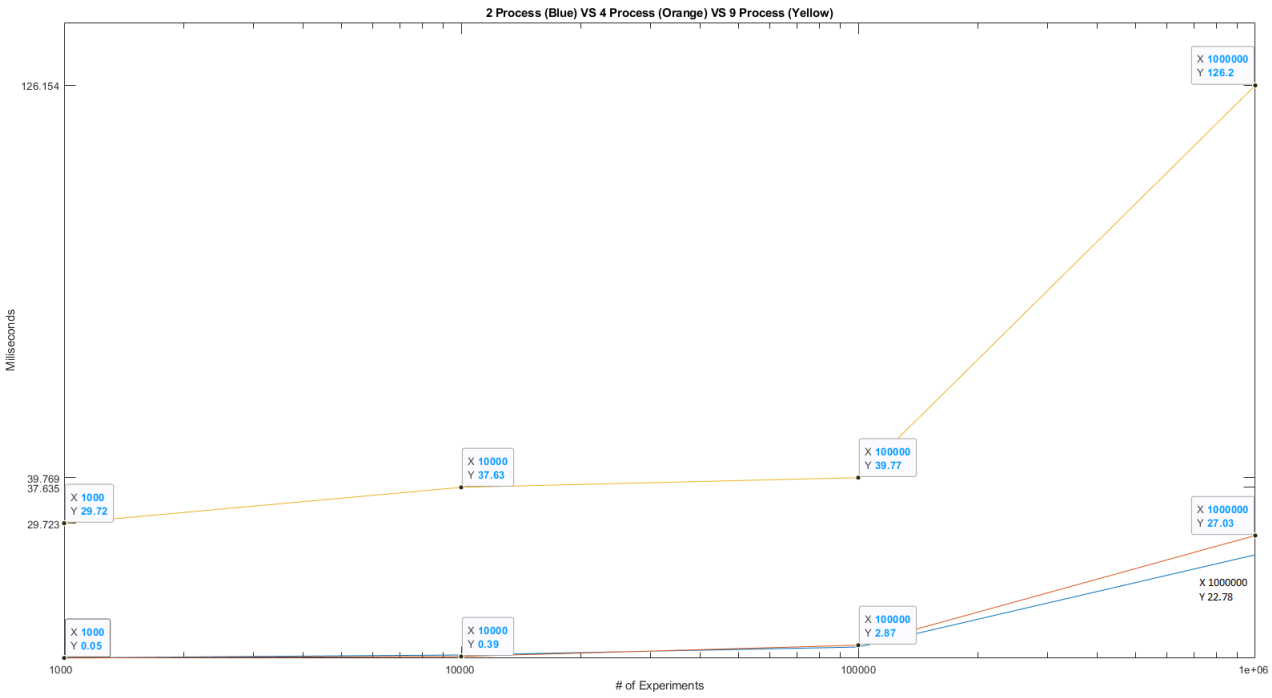

Figure 2: Comparison of 2 (Blue), 4 (Orange) and 9 (Yellow) Number of Processes

As can be seen from Figure 2, 2 and 4 number of processes are very similar in their all tests and 9 process implementation is way slower than them. The reason for this could be the number of created arrays for all processes even if the only master needs them.