Nathan Zacharjak a1704695

Objective

The histogram program produces a histogram outputted to the console made from randomly generated data. It takes a few input parameters from the console, most notable is the number of data points produced, and returns the output histogram using a parallel programming methodology. The goal is to analyse the efficiency of this program and find if there is a minimum data size for which it is useful to parallelize this program on a 4-core system. "Useful" is defined as efficiency E > 0.7 where:

Efficiency = Serial Time / (Number of Processes * Parallel Time)

Methodology

Modifying histogram.c

First, the histogram program was modified to output a number representing the total amount of time the program ran for. The call to the function to output the result was commented out as only 1 core was running this final step, and we are only interested in analysing the parallel part of the code. It also simplified the output of the program making it easy to read the output runtime. After that, the code was modified to add a barrier and a call to read the current time using MPI_Barrier() and MPI_Wtime() for each core. MPI_Wtime() was called again after the parallelised part of the code was run and the difference between these 2 numbers recorded as that core's runtime. The longest runtime out of each of the cores was taken as the total runtime of the program.

Running the tests

The modified program was then compiled and run on a system that had at least 4 cores. The program was given its input parameters though a "params" text file to streamline the testing process. The program was run 3 times on a certain number of cores, with a certain number of data points to generate as given through its last parameter. The minimum of these times was then taken as the final runtime of the program with that many cores and data size, as the minimum time is the least likely to be affected by outside runtime factors than the max time or average time. Constant numbers for the program's 3 other parameters: min data value, max data value and number of bins, were used. 5 Billion data points was decided as the maximum number as errors related to the extreme size of data being processed by the program for larger data sizes began occurring so runtimes could no longer be obtained.

Interpreting efficiency

This runtime was decided to be the "parallel time" in the efficiency formula. The number of processes is the number of cores given to get the parallel time. The "serial time" was taken as the time it took for the program to run on 1 core.

Program execution command

The following command was used to run the program:

```
mpiexec histogram -n [core number] < params > ./perfdata/core[core number]data[data size][test number]
```

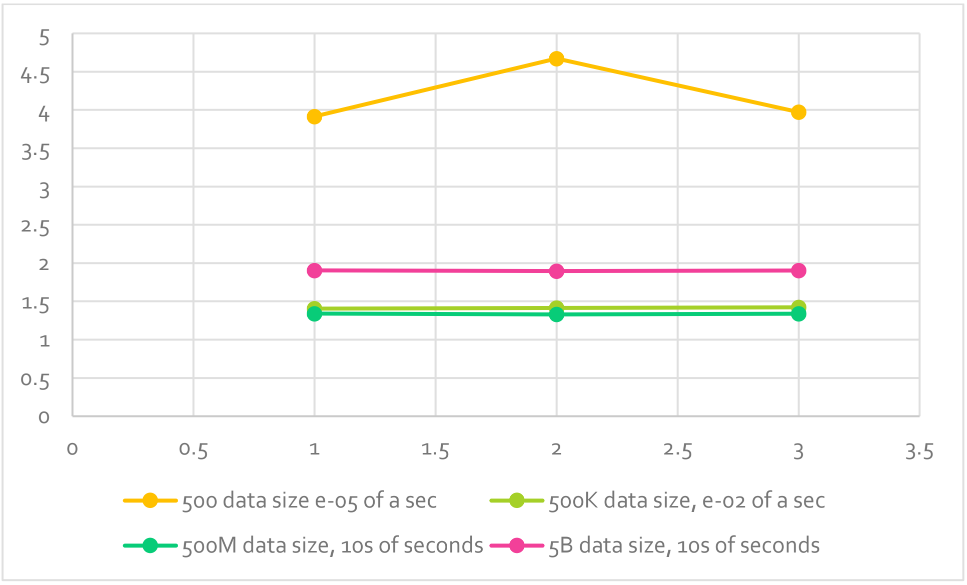
Replacing the bracketed words with their respective numbers.

Nathan Zacharjak a1704695

Results

Runtime in seconds, minimum recorded time:

Core count-> Data size | V	1	2	4
500	0.0000391	0.0000467	0.0000397
500,000	0.00141	0.00141	0.00142
500,000,000	13.4	13.3	13.4
5,000,000,000	19.0	18.9	19.0



Note: one test for 500 data points and 2 cores had an outlier runtime of: 0.000611 seconds, which was ignored.
Also note: Graph lines are in different fractions of a second, but each data point is to the same scale to make the lines comparable, because we're interested in the shape of the graph to see a trend, not the line's actual position on the graph to make each line readable and comparable, else the 500M/5B data points will dominate the graph and the other lines would be flat

Conclusion

Throughout all tests, the program ran for roughly the same amount of time. This gives an efficiency of 0.25, which is less than 0.7 and is clearly showing the program does not perform any differently when run in parallel. Perhaps with a larger number of processors this program will begin to show some signs of parallel efficiency, but even when running the program itself to perform the tests, it was clearly noticeable that the program was taking the same amount of time for each number of cores to run. Unless this program needs more cores to run more efficiently in parallel, from these results, it is not recommended to run the histogram program as a parallel program.