MODULE CODE: CM3103

STUDENT NUMBER: C1545036

PARALLEL PROGRAMMING WITH MPI

Hardware and Software Environment:

CPU:	Intel(R) Core(TM) i7-4790 CPU @3.60GHz
Number of cores:	4
Number of Logical Cores:	8
Operating System:	Ubuntu 18.04.1 LTS
Compiler Version:	gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Computer Name:	rubidium

Timing Experiments:

The first blur sequential non-parallelised code was run first multiple times and the average and standard deviation was taken. It is then followed by compiling and running the parallelised code using MPI on different number of processors denoted by '-n' and the machines used via the 'machines.txt' file. The program is to run processors with number of: 2, 4, 6, 8, 10, 12 each with 10 runs and the average for every processor count is calculated. Below is the command for running the program 10 times. It is repeated for each value of processors.

```
seq 10 | xargs -n1 mpiexec -n 2 -hostfile machines.txt blurMPI
```

Timing Results:

Sequential Code Baseline Average after 10 runs: 0.044143
With Standard deviations of 0.002216

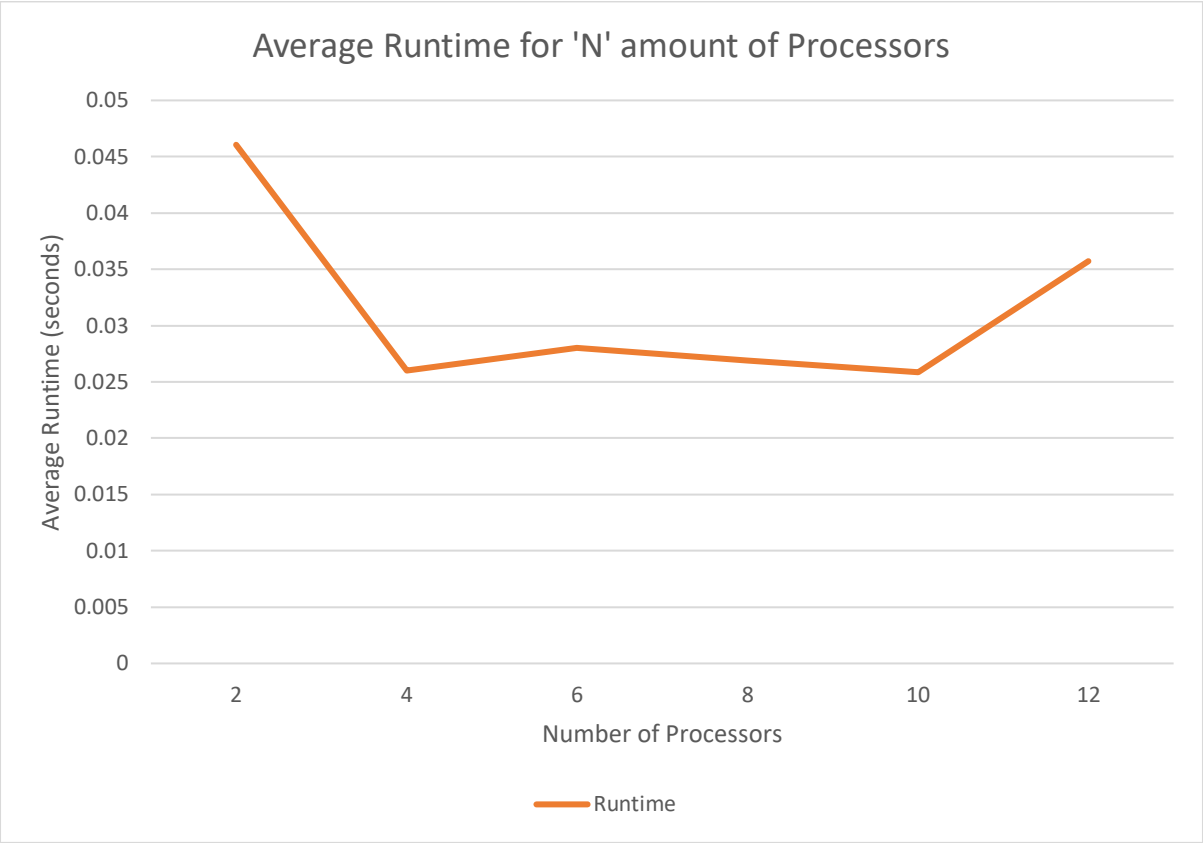

Figure 1. Graph of Average Runtime against Number of Processors.

Results of Average runtimes:

n	2	4	6	8	10	12
Average runtime	0.046057	0.025994	0.028019	0.026869	0.025859	0.035741

Performance Model:

Below is the Speed-up graph derived from the data:

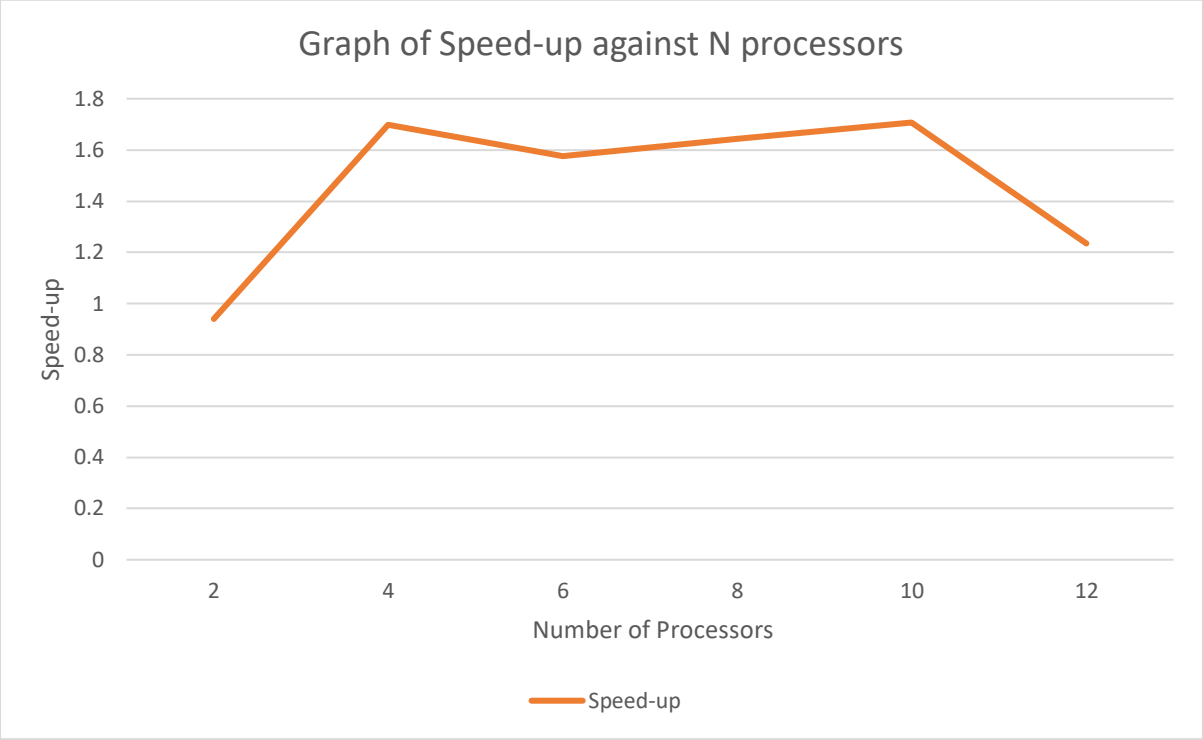

Figure 2. Graph of Speed-up against N processors

The speed-up model was derived by simply taking the Baseline average and divided by the average for n number of processors.
Comparing my speed-up model to the standard speed-up model it shows the same curve, lower number of processors would result in low speed up and as it increases it will peak at a point depending on the size of the problem and will begin to slow down as the number of processors being used exceeds the number processors required to solve the problem. In this case, the speed-up begins to dip beyond 10 processors most likely due to the processors end up taking more time return the results for the blurring of the image.

Discrepancy when using a processor count of 6 and 8 occurred when it dipped in the performance. This could be due to Load imbalances; sizing issues of the problem, as the input file was not a square grid therefore the sudden slow-down could be the processors not working on perfectly scaled problems.

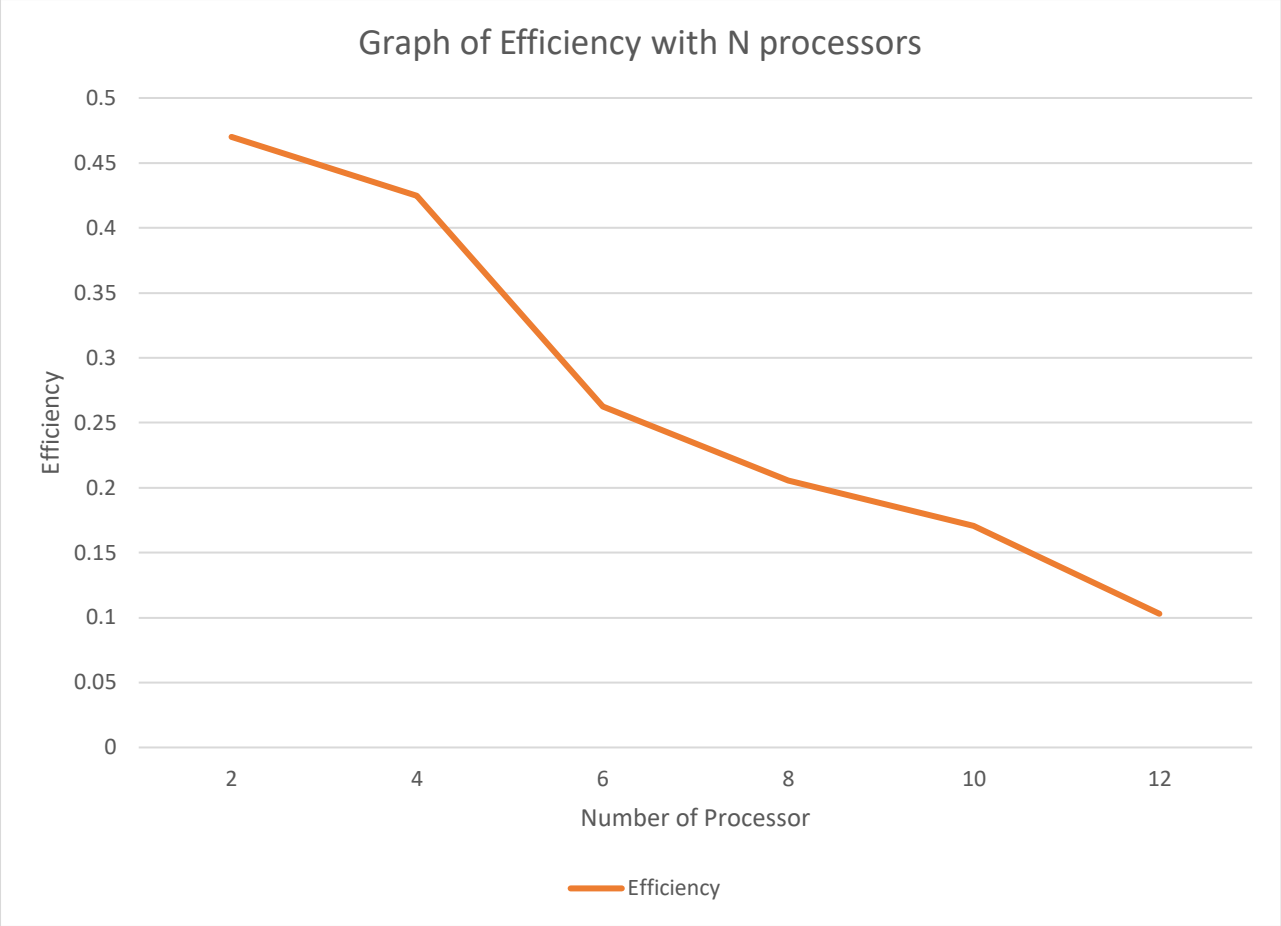

Figure 3. Graph of Efficiency against N processors

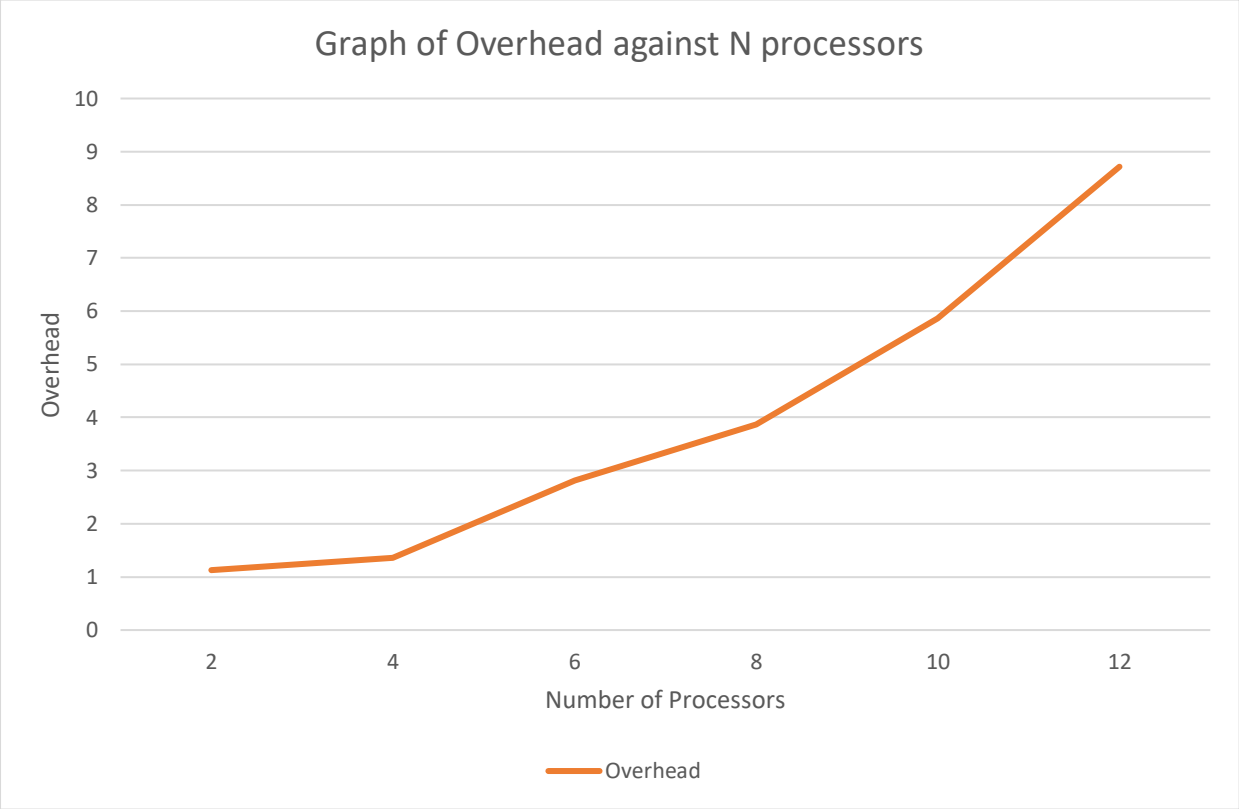

Figure 4. Graph of Overhead against N processors

In terms of the speed up and overhead, it follows the same trend, if you use more processors, the efficiency decreases, and the communication overhead increases exponentially as a result of the decreasing efficiency.

Conclusion:

There may be significant factors like communication overhead that affects the Speed-ups in MPI and also the type of problem may cause certain load imbalances for processors waiting to do work. But the results of the experiment show that as the problem size increases, the speed up also increases with the number of processors. In this experiment, the size of the problem works best with 10 processors as it shows the highest speed up.