MODULE CODE: CM3103

STUDENT NUMBER: C1545036

PROGRAMMING WITH CUDA

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:	Cuda compilation tools, release 9.1, V9.1.85
Computer Name:	rubidium
Graphics Processing Unit:	Nvidia GeForce GTX960

Timing Experiment:

The first blur sequential non-parallelised code was run first multiple times and the average and standard deviation was taken.
blurCUDA.cu is created and tested with the provided image and the outputs are recorded for each phase, during reading the image, Memory allocation, Data Transfer between Host and device memory, Conducting the blurring and outputting blurred image file.
For nblurs = 10,20,40,60,80,160
Each nblurs is ran 10 times and the averages for the values are recorded. The average values for Time to Blur the image is then divided by number of nblurs to obtain the average time taken for each blurring.

Timing Results:

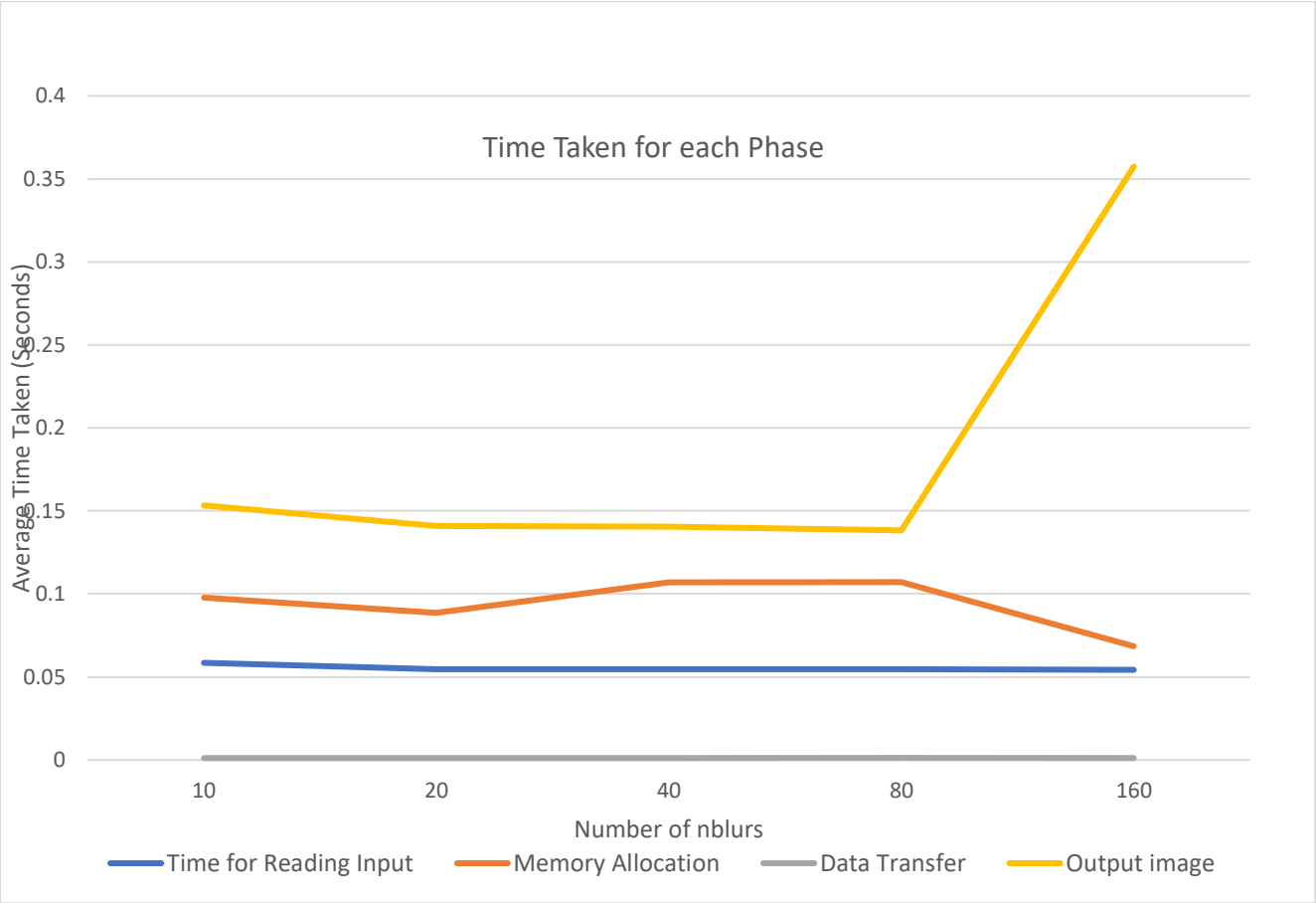

Figure 1. Graph of Time taken for Each Phase

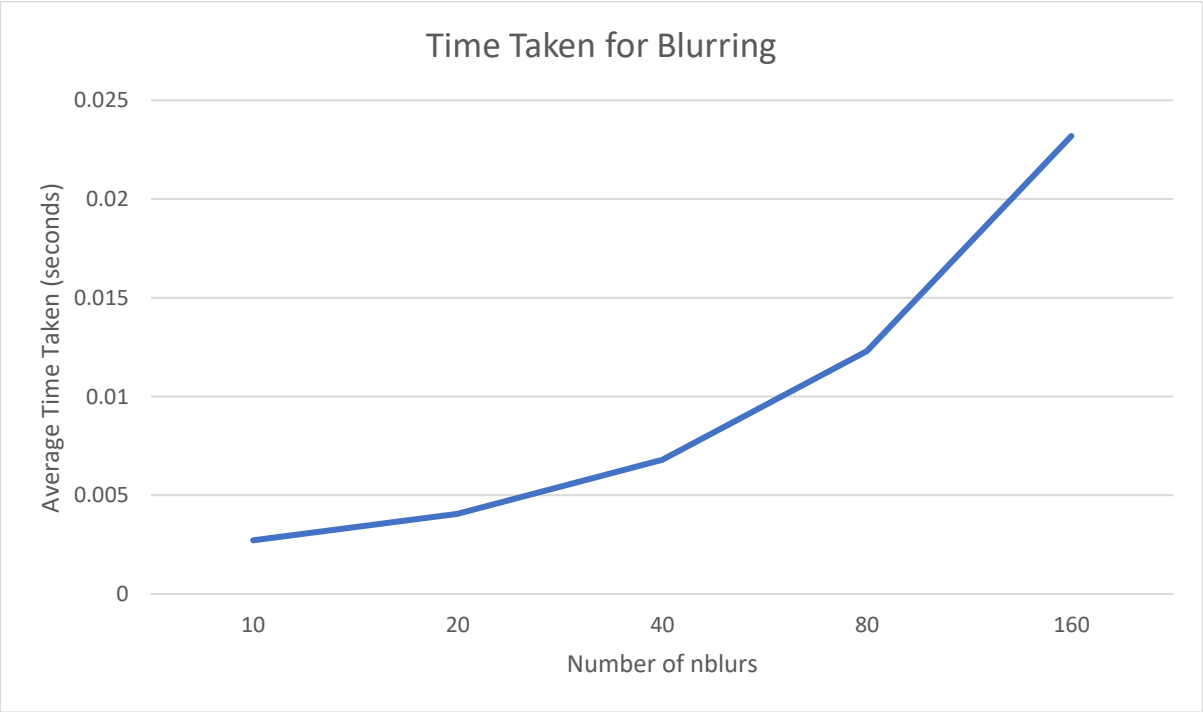

Figure 2. Graph of Time taken for Blurring for each nblurs

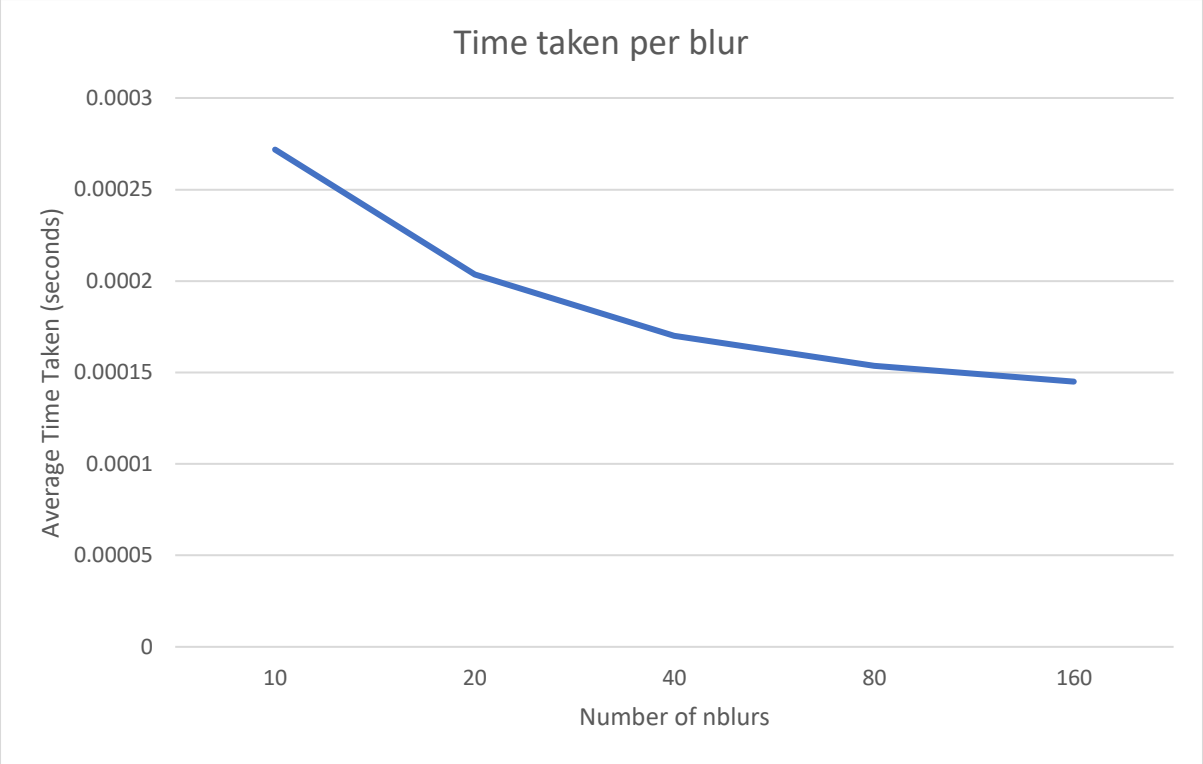

Figure 3. Graph of Time taken for Per-blur for nblurs

Sequential Code Average run-time: 0.044143
Standard deviation: 0.002216

Discussion:

The results from the experiment shows that for the phases of Reading the image file, allocation of memory, transfer of data between host to device, and outputting the blurred image showed consistent values except for the Output phase for nblurs = 160. For nblurs = 160, the sudden increase in time taken to output to file could be due to the image to be outputted being drastically different.

Runtimes for Blurring the image exhibited an exponential curve. As the values of nblurs increases, the time taken also increases exponentially as opposed to linearly. Looking into the average time per blur, as the number of nblurs increase the actual time taken for each blur showing an exponential decline rather than a constant value for time taken to conduct each blurring. The exponential decline instead of a constant resulted in the exponential increase in Figure 2. Possible reason for this might be the graphics processing unit caching similar data and since the blurring process remains the same, the GPU can more quickly reference the needed values.

Possible improvements of the CUDA program could utilise NVIDIA Nsight to determine which parts of the algorithm requires optimization. Another possible method would be to increase the granularity of the data to better exploit parallelism.

Conclusion:

In conclusion, Although the total runtime for the CUDA program took longer than the sequential cod, as the number of nblur iterations increase CUDA becomes significantly faster than the sequential code. Parallelisation offered by CUDA would perform the best when the problem is larger.