C1544657
CM3103
07/12/2017

High Performance Computing

Coursework 3

Programming with CUDA

This report consists of 5 sections. These are:

1) Hardware and Software environment

2) Description of the timing experiments

3) Graphs

4) Discussion of Results

5) Conclusion

Hardware:

Processor: Intel Core i7 -4790 CPU @3.60GHz Quad Core, 8 threads, 8MB Smart Cache

GPU: GeForce GTX 960, 2048MB Total Memory, 128-bit Memory Interface, 1024 CUDA Cores

RAM: 16GB(2x8GB)

Software:

OS: Linux-x86_64

Editor: Sublime text

Programming Language: C

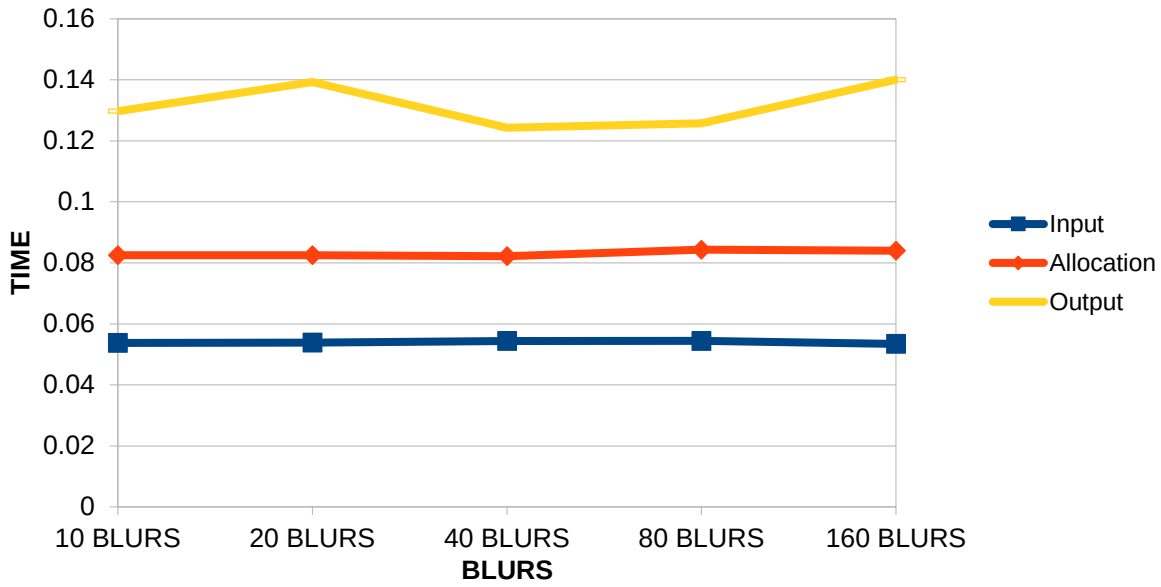
Parallel Computing Platform: CUDA

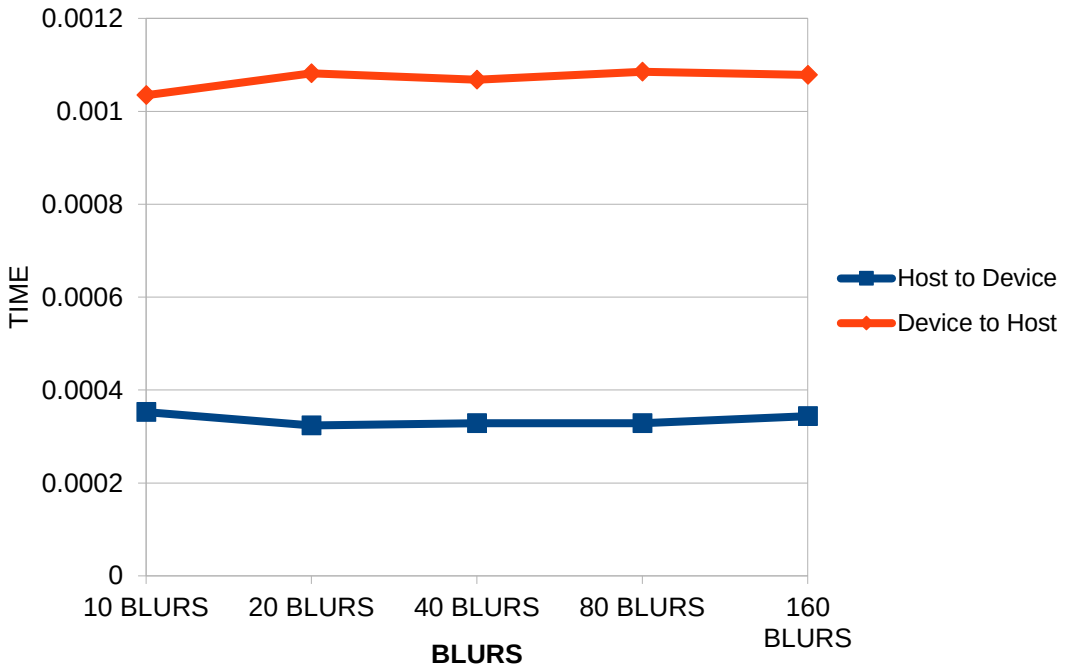
Description of the timing experiments:

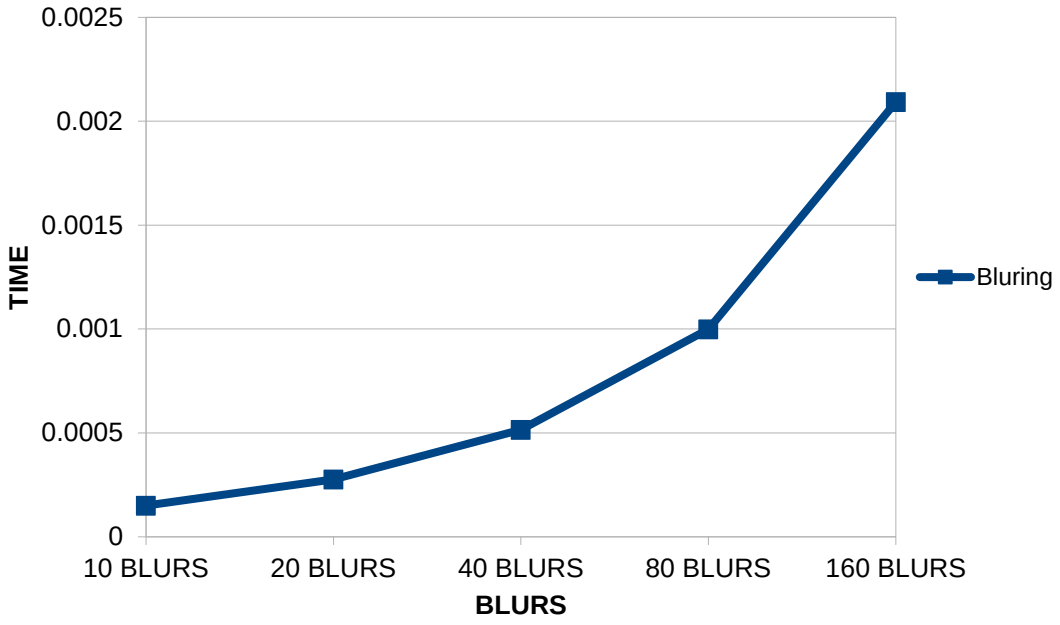
Firstly, I compiled and ran the program 10 times as instructed, finding the average and standard deviation of these 10 times. I then proceeded to produce a parallel CUDA version of the program. The next step was to take measurements of: the time taken to read the input file, allocate device memory, transferring data from host memory to device memory, transferring data from device memory to host memory, doing the blurring and outputting the blurred image. The program was run 10 times for

every different number of blurs. The numbers of blurs used were: 10,20,40,80,160. I used values that double each time to see whether the time for blurring doubles as well. I decided to measure the times for each colour separately, to notice if there was any difference, in the timings, between the colours. After that, I summed the timings of the three colours and produced the average and standard deviation using those sums. The averages and standard deviations of the input and output were calculated as well, and graphs were produced for all of them.

Graphs:







Discussion of Results:

From the first graph, we can see that the time for allocation of device memory remains constant. We can see that because it produces a line very close to a straight horizontal line with minimal differences in time, unaffected by the number of blurs. The same can be observed for reading the input file. This is because reading the input is done in the CPU and has nothing at all to do with the blurs. The work this function does is constant so the time it takes with each number of blurs is also constant. We would expect the same for the outputting the blurred image but instead we get a wavy line. As the values are very small and still very close to each other, we can say that the values are constant.

On the second graph we can see the timings for Transferring data from host to device and from device to host. As we can see, the results produce almost two straight horizontal lines. This means that the times for these procedures are constant and have nothing to do with how many blurs happened. This is because the transfers are always 3 arrays, which do not change size for the number of blurs.

On the third graph we can see how the time for the blurring changes depending on the number of blurs. As expected, the time taken increases as the number of blurs increases. Using doubling values of blurs, I can see that as the number of blurs doubles, the time taken doubles. Dividing the time taken by the number of blurs, I get very similar numbers for every number of blurs. This shows that the time taken per blur is constant. This can also be seen by the curve created on the graph.

By using a function to call the performUpdates for each different colour, I observed that the first time it allocates memory for the Red array, takes much longer than allocating memory on any of the

other colours. My guess for that is that because allocating the memory for the Red array is the first time the CPU communicates with the GPU, it takes some extra time.

Conclusion:

Before this project I had minimal experience with the C programming language and no experience at all with CUDA. This assignment has helped me develop my C programming skills, and made me understand how to use parallel programming using CUDA. I gained the skill to create a code to run in CUDA utilizing both the CPU and GPU of a machine. A very good observation I made during this coursework is that by using the GPU, using CUDA, the time per blur is constant. The time taken for the CPU to communicate to the GPU for the first time is more than expected. Parallelising the program can be good but there are things that we have to account for, such as the time taken for the CPU to communicate to the GPU. So, if the task is very small it would not be wise to use the GPU as we would lose much time in the communication.