Revision to extend this to 100% of the course assessment

COM2039: Coursework

Introduction

The coursework is in five parts. Each of the first four parts involves some additional experimentation on, and thought about, the outcomes of the lab classes in Weeks 3 to 7 inclusive. You will, of course, find it easiest to deliver a good result if you work steadily over the next few weeks and use the remaining time before submission to refine your report.

Each of the first four parts will be worth 10% of this coursework. The fifth part will be worth 60% of the coursework.

Please submit a single pdf report containing your solutions to all five parts.

The front page of the report must include a signed statement confirming that the report is your own work.

This coursework now constitutes 100% of the overall assessment for COM2039.

Your report must be submitted into Surrey Learn by 16:00 on Monday 25th May 2020.

Part 1: Matrix Multiplication

By the end of Lab Classes 2 and 3 you should have two versions of Matrix Multiplication working correctly: one using only global memory on the device; one also using shared memory on the device.

1. To confirm that you have both programs running, copy and paste into your report examples of input matrices A and B, and the resulting matrix C for one run each of both programs. Use the original generator for the input matrices as this uses a random number generator. I want to see the complete matrices for input matrices of order (16, 16). **[2 marks]**

2. Make sure that you have the timing events from Lab Class 3 also set up on your basic matrix multiplication program from Lab Class 2. **[1.5 marks]**

3. Now progressively scale up the matrix size for each of your two implementations and record the execution times for each implementation. Don't forget to keep the row sizes to be multiples of 16 (and use square matrices). It should give you enough to analyse if you follow the sequence of {16, 32, 64, 128, 256, …, 8192}. Provide tables of the results, and also include a plot of how the execution times increase with input matrix size. **[3 marks]**

4. Now discuss the results of this experiment. What is the hypothesis? Do your results confirm or reject the hypothesis? Provide an explanation of the outcome. You may need to refer to online resources such as Stack Overflow to obtain an up to date view on this. **[3.5 marks]**

Part 2: Reduce

In this and the next two parts, when you include code in your report make sure that you comment the code to clearly explain its functionality.

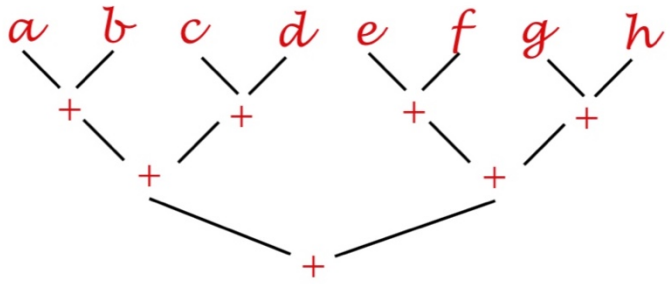
We have seen the basic principle of a parallel algorithm for reducing an array of numbers into a single summary value. I would like you to finalise the coding of the algorithm, generate some timings, and answer some questions about the design of the algorithm.

1. Provide a complete implementation of the algorithm in a form that will Reduce an array of elements whose size will require the use of multiple blocks of threads. I would like you to produce two versions: a variant in which the final reduction of the partial results from each

block is performed on the CPU **[4 marks]**; a second implementation in which the final reduction is performed on the GPU **[2 marks]**. You may assume that the size of the input array is an exact multiple of the block size.

2. Produce sets of timings for each of these two implementations and plot how the respective executions times scale as a function of input array size. Which algorithm is the faster? Explain why you get this result. **[2 marks]**

3. Review the material from the lectures about Warp sizes and thread divergence. Using that, explain why you think the algorithm we used repeatedly partitions the array into two halves instead of using the pattern of data collection that was introduced in Lecture 10 as in the figure below (where a … h represents successive elements in an array). **[2 marks]**



Part 3: Scan

1. Complete the implementation of the Hillis and Steele for the case where multiple blocks are needed in order to handle large input arrays. **[6 marks]**

2. Provide data from a series of runs of your code to show how the execution time scales as input array size increases. **[2 marks]**

3. Review your code and propose two approaches to improving the performance of your implementation. **[2 marks]**

Part 4: Histogram

1. Complete the parallel implementation of histogram using the Kernel that was outlined in Lecture 11. **[4 marks]**

2. Modify the implementation so that instead of atomicAdd() being called on global memory, it is called on partial results in local memory per block (as discussed in the handout for Lab Class 6). **[4 marks]**

3. Provide data from a series of runs of your code to show how the execution time scales as input array size increases for both implementations. Provide an explanation for the differences in performance between the two versions. **[2 marks]**

Part 5: Reflection

For each of the previous four parts, provide a discussion of how the design of the algorithm and any optimisations you have used relate to the basic principles of parallel programming. For example, you will need to discuss how the details of the design have been informed by an

Revision to extend this to 100% of the course assessment

understanding of the physical architecture of the Nvidia GPU including the memory hierarchy. You will find the material in the lectures in weeks 8, 9 and 10 will help you structure these discussions, so please make sure that you are familiar with this content.

Each of the four discussions in this part is worth **[15 marks]**.