DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF SURREY

COM2039 PARALLEL COMPUTING, SEMESTER 2, 2021-22

DR. SUPARNA DE, DEPT. OF COMPUTER SCIENCE

COURSEWORK

Submission deadline: Monday, May 16th (Week 12), 2022, 16:00

General Guidelines

This coursework constitutes 100% of the overall assessment for COM2039. The coursework consists of four questions. Where appropriate the mark carried by an individual part of a question is indicated in square brackets [ ].

The coursework will include the submission and evaluation of the following elements, submitted together as a zipped file:

- coursework report;
- source code for the programming tasks in Question 2 (a-e) and Question 4(a).

The coursework report should be named as follows: report_URN.pdf. The coursework report should contain a page with a declaration of originality as shown on Page 31 of your program handbook. The e-signature can be a scanned picture of your printed signature, a digital signature drawn on computer, or a signature added using Adobe Acrobat's signing feature.

The source code should be submitted as individual Cuda C/C++ source files, corresponding to each programming question, i.e.:
Q2a_URN.cu,
Q2b_URN.cu,
Q2c_URN.cu,
Q2d_URN.cu,
Q2e_URN.cu and
Q4a_URN.cu.

The above six Cuda source files should be submitted together with the report as a **single zipped file**.

Extensions, Late Submissions and Academic Integrity:

- Coursework will be routinely checked for academic misconduct:
- Please refer to your Student Handbook and the advice given on SurreyLearn on plagiarism and collusion and make sure that you understand the regulations.
- If you are in any doubt, please seek advice from your Module Leader or Personal Tutor.
- Both the report and the source code should be submitted electronically via SurreyLearn by the **deadline**. Pay attention to late submission policies of the University: late submissions will lead to 10% penalty per day and any submission after **18th May 2022, 16:00** will receive 100% penalty (i.e., it will not be assessed, and a grade of 0% will automatically awarded).

The assessment will be based on the criteria specified in relevant sections of this document.

Question 1: (30 marks)

Provide the solutions for the following questions, explain your answers for each part, showing the working steps, as appropriate – otherwise you will only get one mark for each correct answer.

(a) What is the ">>" operator? **[4]**
Explain its working in terms of powers of 2. Show its working, with all steps, for the case 12>>2.

(b) Ignoring overheads due to memory access and task communication, calculate the pessimistic and optimistic estimates of overall speedup (up to 2 decimal places) for the case where 75% of the work in a task is sped up by the use of 1000 parallel processors. **[4]**

(c) Assuming that that a kernel called `someKernel` has been defined, how many times will it run based on the following invocation: `someKernel<<<10, 10>>>()` **[3]**

(d) What is the arithmetic intensity of the matrix multiplication kernel using shared memory (Lab class week 3)? **[5]**

(e) Assume that we want to use each thread to calculate two (adjacent) elements of a vector addition. What would be the expression for mapping the thread/block indices to i, the data index of the first element to be processed by a thread? **[4]**

I. i=blockIdx.x*blockDim.x + threadIdx.x +2;
II. i=blockIdx.x*threadIdx.x*2
III. i=(blockIdx.x*blockDim.x + threadIdx.x)*2
IV. i=blockIdx.x*blockDim.x*2 + threadIdx.x

(f) If a CUDA device's SM (streaming multiprocessor) can take up to 1536 threads and up to 4 thread blocks. Which of the following block configuration would result in the most number of threads in the SM? **[4]**

I. 128 threads per block
II. 256 threads per block
III. 512 threads per block
IV. 1024 threads per block

(g) In order to write a kernel that operates on an image of size 400x900 pixels, you would like to assign one thread to each pixel. You would like your thread blocks to be square and to use the maximum number of threads per block possible on the device (your device has maximum number of threads per block as 1024). How would you select the grid dimensions and block dimensions of your kernel? **[6]**

Question 2: (25 marks)

For this question, pls. refer to the relevant Cuda C/C++ (.cu) files included with the coursework. Your goal is to refactor the existing code according to the given instructions.
The modified code files should be renamed according to the convention **Q2x_URN.cu** (where x takes its values ranging from a to e) and should be included in your zipped CW submission on SurreyLearn.

Make sure that your solution programs:

- compile and run on the lab machines, without requiring any modifications

- are neatly formatted and include comments to explain what you have done.
- include comments that refer to the technical concepts covered in the lectures and accompany each modification made in the given working code.

Complete and appropriate comments account for 50% of the marks for each individual sub-question (a-e).

(a) The code in Q2a.cu contains two functions, both with print `"Hello from the CPU"` messages. Refactor the `helloGPU` function in the source file so that it actually runs on the GPU, and prints a message indicating that it does. Also, ensure that `Hello from the GPU` prints **before** `Hello from the CPU`. **[5]**

(b) The program in Q2b.cu contains a working kernel that is printing a failure message. Update the execution configuration so that the success message will print. **[5]**

(c) Currently, the loop function inside the program in Q2c.cu runs a `for` loop that will serially print the numbers `0` through `9`. Refactor the loop function to be a CUDA kernel which will launch to execute `N` iterations in parallel. After successfully refactoring, the numbers `0` through `9` should still be printed. When refactoring the loop to launch as a kernel, use the execution configuration to control the number of iterations, using *only 1 block of threads*. **[5]**

(d) For the program in Q2d.cu, refactor the code the `loop` function to be a CUDA kernel which will launch to execute `N` iterations in parallel. After successfully refactoring, the numbers `0` through `9` should still be printed. For this question, as an additional constraint, use an execution configuration that launches *at least 2 blocks of threads.* **[5]**

(e) Currently the program in Q2e.cu will not work; refactor the code so that it works correctly and meets the following conditions:
- `a` should be available to both host and device code.
- The memory at `a` should be correctly freed. **[5]**

Question 3: Task Dependency Graphs (10 marks)

For the 2 task dependency graphs shown in Fig. Q3 (a) and (b), calculate the following:

I. Maximum degree of concurrency
II. Critical path length
III. Average degree of concurrency (express this as a ratio)

Assume that the nodes in the graph are equally weighted in terms of the amount of work required by the corresponding task. Include explanations for your answers.

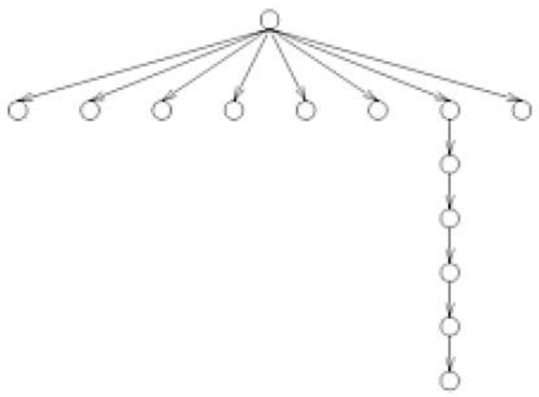
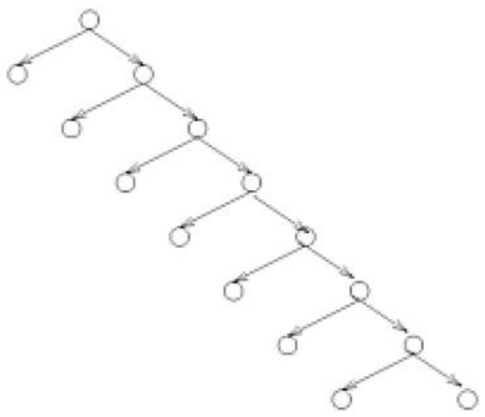

Fig. Q3. Task dependency graph (a) *(b)*

Question 4: Reduce (35 marks)

(a) In this question, when you include code in your report make sure that you comment the code to clearly explain its functionality. The developed code should be included in your zipped CW submission on SurreyLearn and be executable on the lab machines without requiring any modification.

Provide a complete implementation of a parallel algorithm that will Reduce a 1D array of elements into a single summary value, where the size of the input array will require the use of ***multiple blocks of threads***. The reduction should give the sum of the list. **[20]**

Additionally, the code solution should meet the following **requirements/assumptions**:

- your kernel should be able to handle input lists of arbitrary length. However, for simplicity, you can assume that the input list will be at most 65535 elements. The kernel code should be written so that it can handle the above stated maximum input array by only one kernel launch. Further assume that the reduction sums of each section generated by individual blocks will be summed up by the CPU.
- the code should, as a minimum, be tested to output the correct reduced value for the following InputArraySize values:
  - InputArraySize: 8
  - InputArraySize: 1024
  - InputArraySize: 1025
  - InputArraySize: 65535
- assume that the input array is formed of elements whose value corresponds to the index of the array, with the last element value corresponding to (InputArraySize-1), i.e. for an array of InputArraySize: 8, the elements in the input array would be [0, 1, 2, 3, 4, 5, 6, 7].
- inclusion of Cuda error handling and event specification at appropriate points in the program and minimum diagnostic print statements
- explanatory comments for design decisions and to explain the functionality
- include instructions at the top of the .cu file (as comments) how the above different InputArraySize values can be specified for different runs of the program.

Marking criteria for this coding task:
Code build without errors that meets the above outlined requirements: 10 marks

Appropriate comments and code quality: 10 marks

(b) How many global memory reads and writes are performed by your kernel? Explain. **[6]**

(c) Describe what optimizations were performed to your kernel to achieve a performance speedup. **[4]**

(d) Describe what further optimizations can be implemented to your kernel to achieve a performance speedup. **[5]**