SCIT

School of Computing & Information Technology

CSCI376 – Multicore and GPU Programming
Spring 2019

Assignment 2

Due on Monday, 23rd September 2019 at 09:00

OpenCL vector datatypes and built-in functions should be used wherever possible.

Part 1: Basic Kernel Programming

Task:

Write a program that does the following:

- In the host program, create two arrays (or STL vectors) as follows:
  - Array 1: An 8 element array of ints with random values between 0 and 9.
  - Array 2: A 16 element array of ints. Initialise the first half of the array with values from 1 to 8 and the second half with values from -8 to -1.

(0.5 marks)

- Write a kernel that
  - Accepts array 1 as an array of int4s, array 2 and an output array
  - Reads the contents from array 1 and 2 into local memory
    - Copy the contents of array 1 into an int8 vector called v
    - Copy the contents of array 2 into two int8 vectors called *v1* and *v2* (using **vload*n***)
  - Creates an int8 vector in private memory called *results*. The contents of this vector should be filled as follows:
    - Check whether **any** of the elements in *v* are greater than 5
      - If there are, then for elements that are greater than 5, copy the corresponding elements from *v1*; for elements less than or equal to 5, copy the elements from *v2* (using **select**).
      - If not, fill the first 4 elements with the contents from the first 4 elements of *v1* and the next 4 elements with contents from the first 4 elements of *v2*
  - Stores the contents of *v*, *v1*, *v2* and *results* in the output array (using **vstore*n***)

(2 marks)

- In the host program, check that the results are correct and display the contents of the output array (0.5 marks)

Part 2: Shift Cipher

A shift cipher (a.k.a. Caesar's cipher) is a simple substitution cipher in which each letter in the plaintext is replaced with another letter that is located a certain number, *n*, positions away in the alphabet. The value of *n* can be positive or negative. For positive values, replace letters with letters located *n* places on its right (i.e. 'shifted' by *n* positions to the right). For negative values, replace letters with letters located *n* places on its left. If it reaches the end/start of the alphabets, wrap around to the start/end.

For example:

If *n* = -3, each letter in the plaintext is replaced with a letter 3 positions before that letter in the alphabet list.

Plaintext:	**The quick brown fox jumps over the lazy dog.**
Ciphertext:	**QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD.**

Note that in the example above, c → Z, since 3 positions before 'c' wraps around to the end of the alphabet list and continues from 'Z'. Similarly, a → X and b → Y.

Leave anything that is not an alphabet as is (i.e. punctuations and spaces).[1]

Decrypting the ciphertext is simply a matter of reversing the shift.

Task:

a. Write a normal program that reads the contents from a text file (a test file called "plaintext.txt" has been provided. Note that your program is to work with any text file). The program should prompt the user to input a valid *n* value, then encrypt the plaintext using the shift cipher method described above, and output the ciphertext into an output text file. To ensure that the encryption was performed correctly, your program must also decrypt the ciphertext to check whether it matches the original plaintext (albeit in upper case). (2 marks)

b. Write an OpenCL program to perform encryption, and decryption in parallel. Check that the resulting ciphertext matches the output that you obtained from (a), and output the ciphertext into an output text file. The program must also be able to accept a text file containing the ciphertext and perform decryption in parallel. (3 marks)

c. Extend your program so that the user can perform parallel encryption, and decryption, by substituting characters based on the following lookup table:

[1] Note that to avoid leaking information (e.g., word length), by contention the ciphertext is usually converted to upper case letters that are organised in groups of five-letter blocks and anything that is not a letter is removed. However, there is no necessity to do this in the assignment.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
G	X	S	Q	F	A	R	O	W	B	L	M	T	H	C	V	P	N	Z	U	I	E	Y	D	K	J

Based on the table above, for encryption the letter a (or A) will be replaced by G, b (or B) will be replaced by X, c (or C) will be replaced by S, etc.

(2 marks)

Part 3: Brute-force Parallel Passcode Cracking

Given the following information about a passcode:

- The passcode contains 8 digits (0-9)

Design and implement an efficient parallel program using OpenCL to perform brute-force passcode cracking. Note that your program design must attempt to match the **entire** passcode (i.e. you are NOT allowed to find the digits one-by-one. This means that your program cannot try to find the 1st digit, followed by the 2nd digit, etc.)

Task:

- Store the *passcode* as a global variable at the top of your .cl file. For the passcode, use your UOW student ID number with a 7 in front (i.e. 8 digits in total).
- Write a function in the .cl file (not a kernel function) called *checkPasscode*
  - This function should accept **entire** passcode attempts
    - NOTE: This function is to check whether all digits in the attempts match the stored *passcode*
  - Check whether any attempts match the stored *passcode*
  - Returns the correct *passcode* if a match is found

(1 mark)

- Your program should search through the space of all possible 8-digit passcodes to find a match with the stored passcode
  - It should contain an OpenCL kernel that will call the *checkPasscode* function
  - The host code is to display information about the work-item that found the passcode match (i.e. its global id, work-group id and local id)

(4 marks)

Instructions and Assessment

Organise your solutions into 3 folders (named Part1, Part2 and Part3) and put all your **source files** (source files only, i.e. .cpp, .h and .cl files) into the respective folders.

Zip the 3 folders a single file and submit this via Moodle by the due date and time (do **NOT** zip entire visual studio project files as this can be very large). Assignments that are not submitted on Moodle will not be marked.

You will have to demonstrate your working program during the lab in Week 9. You must be ready for this lab task to be assessed at the start of this lab. Do not try to fix your code during this lab, otherwise late penalties may apply. Your program will be marked in the lab so it must work on the computers in the lab (or you must demonstrate it in the lab on your own laptop).

The assignment must be your own work. If asked, you must be able to explain what you did and how you did it. Marks will be deducted if you cannot correctly explain your code.

NOTE: The marking allocations shown above are merely a guide. Marks will be awarded based on the overall quality of your work. Marks may be deducted for other reasons, e.g. if your code is too messy or inefficient, if you cannot correctly explain your code, etc.

For code that does not compile, does not work or for programs that crash, the most you can get is half the marks.