SCIT

School of Computing & Information Technology

CSCI376 – Multicore and GPU Programming
Spring 2020

Assignment 2

Due on Friday, 16th October 2020 at 17:00

Task 1 – Basic Kernel Programming

Task 1a

Write a program using OpenCL that uses a kernel to fill in the contents of an array of 512 numbers in parallel. The program is to prompt the user to enter a number between 1 and 100 (inclusive). The program is to check whether the user entered a valid number, if not, the program should quit. If a valid number was entered, enqueue a kernel (using the enqueueNDRangeKernel function) that accepts the number and an array, and fills in the contents of the array using the number (and the work-items' global ID) as follows:

- If the user enters 1, the resulting contents of the array should be: 3, 4, 5, 6, 7,… until 514
- If the user enters 2, the resulting contents of the array should be: 3, 5, 7, 9, 11,… until 1025
- If the user enters 3, the resulting contents of the array should be: 3, 6, 9, 12, 15,… until 1536
- …
- If the user enters 100, the resulting contents of the array should be: 3, 103, 203, 303, 403,… until 51103

After kernel execution, display the contents of the output array on screen.

(2 marks)

Task 1b

Write a program using OpenCL to do the following:

- Host code.
  - Create two STL vectors, named *vec1* and *vec2*. Initialise their contents as follows:
    - *vec1*: A vector of ints that contains 32 elements. Initialise the elements with random values between 10 and 20.
    - *vec2*: A vector of ints that contains 16 elements. Initialise the first half of the vector with values from 2 to 9 and the second half with values from -9 to -2.
  - Create and initialise the necessary memory objects to pass *vec1* and *vec2* as input to the kernel, and another memory object for kernel output.

- Enqueue the kernel such that each work-item will process 8-elements from *vec1*, respectively, i.e. work-item 1 will refer to elements 0-7, work-item 2 will refer to elements 8-15, etc.
- After kernel execution, obtain the output from the kernel and display the results on screen.

(2 marks)

- Kernel code.
  - The kernel is to have three parameters.
    - *input1*: an input array of type int4
    - *input2*: an input array of type int
    - *output*: an output array of type int
  - When the kernel is enqueued by the host, the contents of *vec1* and *vec2* are to be passed to the kernel as *input1* and *input2*, respectively.
  - In the kernel, copy the contents from *input1* and *input2* into three private OpenCL vectors of type int8:
    - *v*: is to contain 8 elements obtained from *input1*. The content of *v* will vary based on the work-item, such that
      - for work-time 1, *v* will contain vectors 0-1 from *input1*

        (Note: the contents of two int4 vectors, gives a total of 8 elements)
      - for work-time 2, *v* will contain vectors 2-3 from *input1*
      - etc.
    - *v1*: is to contain elements 0-7 from *input2* (use the **vload*n*** function)
    - *v2*: is to contain elements 8-15 from *input2* (use the **vload*n*** function)
  - Create an int8 vector, named *results*. The contents of this vector should be filled as follows:
    - Check whether any of the elements in *v* are greater than 17
      - If there are, then for elements in *v* that are greater than 17, copy the corresponding elements from *v1*; for elements less than or equal to 17, copy the corresponding elements from *v2*. (use the **select** function)
      - Otherwise, 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 (use the **vstore*n*** function). Note that each work-item will have to store its respective results in the output array.

(3 marks)

Example output

```
Work-item 0
v        : 18 19 19 11 17 11 20 10
v1       :  2  3  4  5  6  7  8  9
v2       : -9 -8 -7 -6 -5 -4 -3 -2
results  :  2  3  4 -6 -5 -4  8 -2

Work-item 1
v        : 11 10 15 16 17 13 14 12
v1       :  2  3  4  5  6  7  8  9
v2       : -9 -8 -7 -6 -5 -4 -3 -2
results  :  2  3  4  5 -9 -8 -7 -6

Work-item 2
...
```

Task 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.**
Decrypted:	**THE 'QUICK' BROWN FOX JUMPS OVER THE "LAZY" DOG.**

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 should remain in the encrypted/decrypted text without alteration).[1] The encrypted/decrypted text should all be in upper case.

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

[1] Note that to avoid leaking information (e.g., word length), by convention 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, for this assignment, DO NOT remove anything that is not a letter and DO NOT organise in groups of five-letters.

Task 2a

Write a normal C/C++ program (not using OpenCL) that reads the contents from a text file called "plaintext.txt" (a test file has been provided). 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 called "ciphertext.txt". To ensure that the encryption was performed correctly, your program must also decrypt the ciphertext into a file called "decrypted.txt" to check whether it matches the original plaintext (albeit in upper case).

(2 marks)

Task 2b

Write an OpenCL program to perform the same functionality as in Task 2a, but in parallel. Note that it is more efficient to use OpenCL vector data types for processing in the kernel, as less work-items will be required and elements in a vector can be processed simultaneously.

(4 marks)

Task 2c

Write an OpenCL program to perform parallel encryption, and decryption, by substituting characters based on the following lookup table:

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. As with Task 2b, it is more efficient to use OpenCL vector data types for processing in the kernel.

(2 marks)

Screenshots

For **each task**, include screenshots with your submission to demonstrate that the programs work on your computer. The screenshots should capture the input (if any) and the output of your programs. Use one of the common image formats (i.e. jpg/png/bmp).

Instructions and Assessment

Organise your solutions into 2 folders, named Task1 and Task2 (you may include subfolders as necessary), and put all your **source files** (i.e. files required to compile and run your program, e.g., .cpp, .h and .cl files) into the respective folders.

Zip the 2 folders a single file and submit it via Moodle by the due date and time (Please do not use the .rar format, and do **NOT** zip entire visual studio project folders as this can be very large). Assignments that are not submitted on Moodle will not be marked.

You may be asked to demonstrate your program on your computer. 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. Hence, it is better to comment out sections of code that do not work, and provide a note for the marker in a readme.txt file.