University of Wolverhampton
Faculty of Science and Engineering
Department of Mathematics and Computer Science

Module Assessment

Module	6CS005 – High Performance Computing
Module Leader	Pooja Kaur
Semester	1
Year	2025/26
Assessment	Portfolio
% of module mark	100%
Due Date	Date will be published on Canvas
Hand-in – what?	**Portfolio as specified in this document**
Hand-in- where?	Canvas
Pass mark	40%
Method of retrieval	Submit the resit assessment (will be distributed at the end of the module) by end of resit week (July)
Feedback	Individual feedback via Canvas, in addition verbal feedback is available in class.
Collection of marked work	N/A\

Learning Outcomes

- LO1 Demonstrate and apply knowledge and understanding of: the essential facts, concepts, principles, theories and practices enabling graduate employment in High Performance Computing.
- LO2 Apply appropriate theory, tools and techniques to the development of software for High Performance Computing.

Assessment overview

This portfolio is split up into 4 separate tasks which will test your knowledge of advanced multithreading and GPGPU programming using CUDA. Each task should be zipped up into a single zip folder containing all C/CUDA and resource files for the submission on Canvas.

1. **Word Occurrence Counting using Multithreading (10% - 100 marks) – Pthread**

   In this task, you will be asked to use the Pthread library to count the occurrence of words within a single text file by dividing the workload across multiple threads. Your programme should read through the file and calculate how many times each word appears. The number of threads to be used should be specified by the user through the command line as well as the name of the file. Each thread should be responsible for processing a distinct portion of the data, with dynamic slicing ensuring the workload is balanced evenly. Once all threads have completed their tasks, the results should be merged and written into a separate output file (e.g. result.txt), with each

word displayed alongside its frequency of occurrence. The final output can be sorted alphabetically or by frequency depending on your choice.

Counts word occurrences from a file using multithreading and dynamic slicing based on thread count (50 marks)

Uses proper synchronisation to manage shared data safely (30 marks)

Produces correct and well-formatted output with word frequencies (20 marks)

2. Multiple operations with Matrices using multithreading (20% - 100 marks) - OpenMP

You will create a matrices operations program in C that uses multithreading with OpenMP. Matrices are two-dimensional arrays; for this application you will read them from a single text file supplied on the command line and perform a set of operations in parallel. Use command-line arguments to specify the input file path (argv[1]) and the number of threads to use (argv[2]). The input file contains multiple matrices in sequence, and you must process them as pairs in order (matrix 1 with matrix 2, matrix 3 with matrix 4, …). Store every matrix using dynamic memory allocation (malloc). For each pair (A, B), attempt:

- addition (A+B)
- subtraction (A−B)
- element-by-element multiplication (A.*B) (each element of A * each element of B)
- element-by-element division (A./B) (each element of A / each element of B)
- transpose of A (A^T) (when you swap rows and columns, so rows become columns, and columns become rows)
- transpose of B (B^T) (when you swap rows and columns, so rows become columns, and columns become rows)
- matrix multiplication (A×B)

Use OpenMP to parallelise the computations and split the work across n threads; if the user requests more threads than the largest relevant dimension for that operation (e.g., number of rows of the result or max of the pair's dimensions), cap the actual threads to that maximum. Your program must handle any matrix sizes found in the file and decide which operations are valid based on dimensions:

- if A and B have the same shape, addition, subtraction, and element-wise ops are valid;
- if A.cols == B.rows, then A×B is valid;
- A^T and B^T are always valid.
- If an operation cannot be done, print a clear message for that operation (e.g., "Addition cannot be done (shapes differ).") to the results file and continue with the rest of the operations for that pair;
- for element-wise division, if shapes match but a divisor entry is zero, write NaN at that position and still complete the operation.

All outputs should be written either to a single file named results.txt (containing the results of all pairs, in order, with each pair clearly separated), or to separate files (for example, results_pair1.txt, results_pair2.txt, …) with each file containing the full set of results for that specific pair. Whichever approach is chosen, the output must be consistent and readable, with a clear format that includes the operation name and the result size (for example, Addition - 3,3)),

and then the matrix values (comma-separated, one row per line). For operations that are not applicable, only a one-line message should be printed (for example, “Addition cannot be done – different size”). If a pair is not multipliable (e.g., 3×3 with 2×2), still output addition/subtraction/element-wise/transpose results as applicable.

if a pair is multipliable (e.g., 2×3 with 3×2), output the product but do not attempt addition/subtraction/element-wise ops when sizes are not the same.

Validate parsing thoroughly (missing file, invalid header, wrong row counts, non-numeric tokens) and print clear errors while cleaning up memory. As a minimum, use the standard C file functions fopen(), fclose(), fscanf(), and fprintf() for all I/O; stdin/stdout redirection is not acceptable. Free all allocated memory before exit.

Read data from file appropriately (20 marks)

Using dynamic memory (malloc) for matrix A and matrix B (10 marks)

Creating an algorithm to *add, subtract, multiply (A×B), transpose (A^T/B^T), and element-by-element multiply/divide correctly (20 marks)

Using OpenMP multithreading with balanced work (30 marks)

Storing the correct output matrices in one or multiple result file/s (20 marks).

3. Password Cracking with Files using CUDA (30% -100 marks)

For this task you are required to crack passwords using CUDA from a file. The programme should read a file of encrypted passwords (one per line) and use the GPU to recover the original passwords. A decryption function will be provided and must be converted to a CUDA kernel function. The file name must be passed through the command line (argv). As you are reading in a file of data, your blocks and threads should be dynamically split accordingly based on the number of passwords. For example, if the file contains 10,000 lines, an appropriate split would be 10 blocks and 1000 threads or 100 blocks and 100 threads. For this assessment, you will not be “required” to use blocks and threads in the y and z axis however, at the minimum, you should be using multiple blocks and threads in the x. For example, you should not be using 1 block and 1000 threads (max threads per block) and then doing 10 computations in each thread if the total computation count is 10,000, you should aim to make the threads do as minimal work as possible. Each encrypted password in the file must be processed, and the decrypted passwords returned to the host and written to an output file (for example results.txt or decrypted.txt). GPU memory must be allocated correctly, checked for errors, and freed once the task is complete.

Read in file to the CPU and stored in memory appropriately (5 marks)

Allocate GPU memory according to size of text file and transfer data to GPU memory (20 marks)

Passwords decrypted via the kernel function (35 marks)

Decrypted passwords transferred safely back to the CPU (Host) (20 marks)

Write decrypted passwords to a file (15 marks)

Memory deallocation – freeing memory appropriately (5 marks)

4. Sobel Edge Detection using Cuda across many PNG images (40% - 100 marks)

For this task, you are required to apply a well-known edge detection technique known as the Sobel edge detection algorithm. Sobel edge detection uses a convolution process consisting of a minimum of two kernels, commonly 3x3 matrices or kernels. One is called **Gx** and measures changes from left to right (the horizontal gradient). The other is called **Gy** and measures changes from top to bottom (the vertical gradient). You may see different versions of Gx and Gy on the internet. This is because sometimes the signs are swapped, or the kernel is flipped. Do not worry about this, when you combine Gx and Gy the result is the same.

How does convolution work using a 3x3 original image and the Gx kernel:

-1	-2	-1
0	0	0
1	2	1

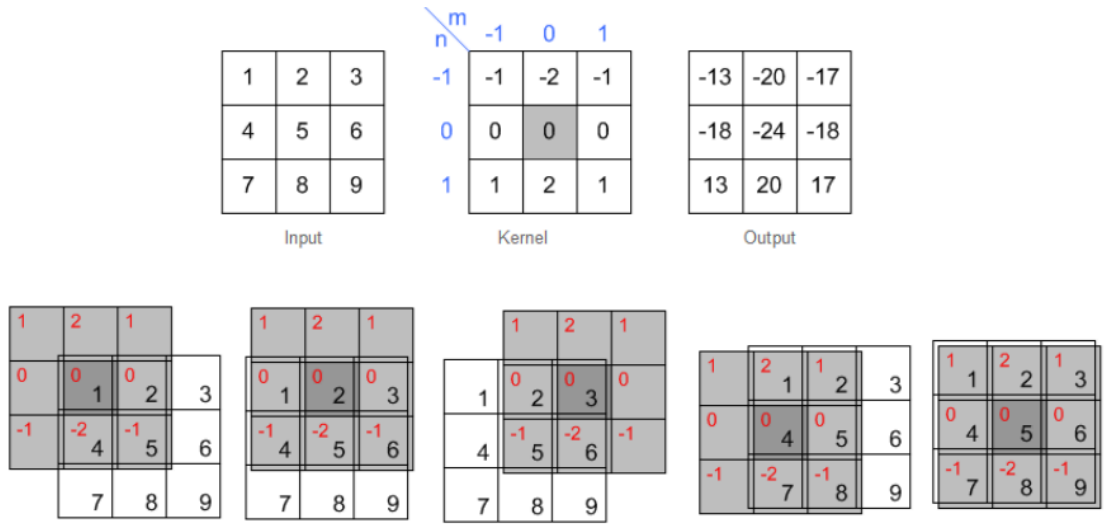
Gx

-1	0	1
-2	0	2
-1	0	1

Gy

Convolution works by sliding the kernel across the image one pixel at a time. For each pixel, you take its 3×3 neighbourhood, place the kernel on top, and multiply each pixel value by the matching kernel value. You then add all these numbers together. This gives you a single result for that pixel. Doing this with the Gx kernel gives the horizontal gradient, while doing the same with the Gy kernel gives the vertical gradient. These two values are then combined to give the overall edge magnitude at that pixel.

A common problem occurs at the borders of the image, where there are not enough neighbours to fill the 3×3 patch. To solve this, we use a technique called zero padding. This means we imagine that the space outside the image is filled with zeros. Zero padding allows the kernel to be applied to every pixel, including those at the edges, without changing the size of the output image.



Process (Convolution) in a simple way:

a. Place the kernel on top of a section of the input image.
b. Multiply each overlapping number by the kernel value.
c. Add all results together → this gives **one number** in the output.
d. Slide the kernel across the whole image, repeating this for every pixel.

Once you've calculated Gx and Gy results for every pixel, you combine them to find the overall edge strength at each position. This is done using a formula called the gradient magnitude:

$$G = \sqrt{Gx^2 + Gy^2}$$

This means you square the Gx result, square the Gy result, add them, and then take the square root. The final number represents how strong the edge is at that pixel.

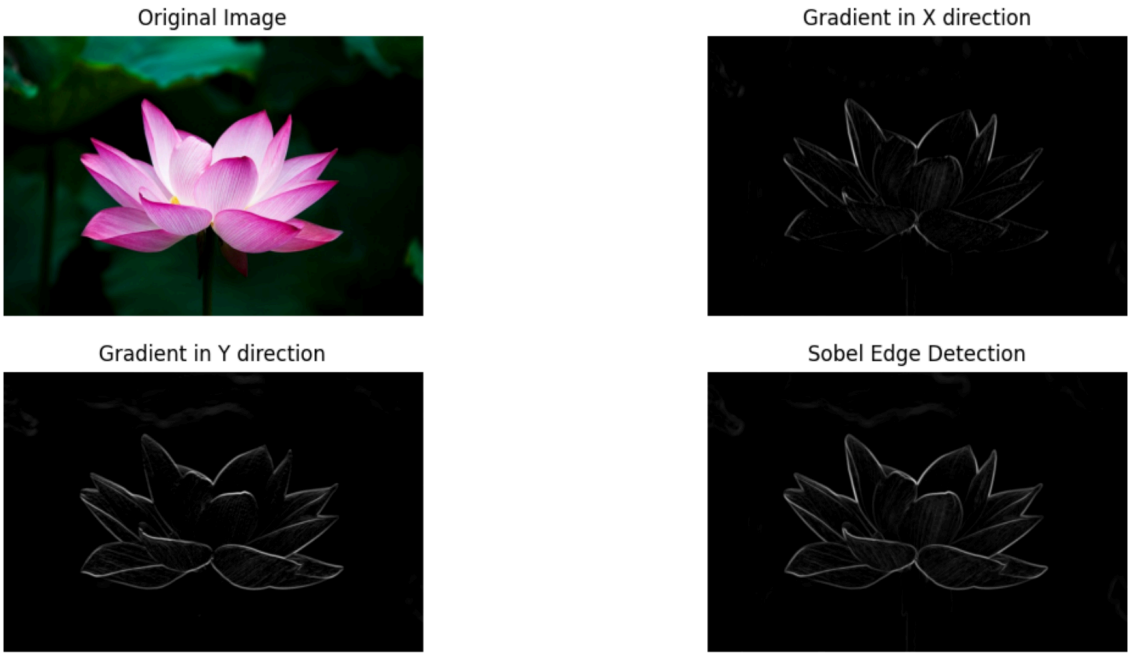
Let's take a simple example. Suppose that, after applying the Gx kernel to a particular pixel, you get a value of 6. After applying the Gy kernel to that same pixel, you get a value of 8. These numbers are not from the kernel itself, but the **results** after applying the kernel to the image.

- Gx = 6
- Gy = 8

Then the edge magnitude (G) is:

$$\text{Magnitude} = \sqrt{6^2 + 8^2} = \sqrt{36 + 64} = \sqrt{100} = 10$$

So, the edge strength at that pixel is now 10. Repeat this process for every pixel in the image and you will produce a new image, called an **edge map,** that highlights the edges in the original image. Bright areas in the output show strong edges, and dark areas show flat regions.



Read image(s) into CPU memory and store appropriately – (5 marks)

Allocate GPU memory according to image size and transfer data to GPU – (20 marks)

Edges detected correctly via the CUDA kernel function – (35 marks)

Edge image transferred safely back to CPU (host) – (20 marks)

Write edge-detected output image(s) to file – (15 marks)

Memory deallocation – freeing both GPU and CPU memory – (5 marks)

Important Message

You may be asked to clarify your assessment after moderation has taken place. This is to ensure the work has been completed by the student.

You must achieve 40 percent overall to pass this module. There will be a resit opportunity during resit week (July) to achieve a pass.

Submission of work

Your completed work for assignments must be handed in on or before the due date. ***You must keep a copy or backup of any assessed work that you submit. Failure to do so may result in your having to repeat that piece of work.***

Penalties for late submission of coursework
Standard Faculty of Science and Technology arrangements apply.
ANY late submission (without valid cause) will result in 0 marks being allocated to the coursework.

Procedure for requesting extensions
If you have a valid reason for requiring an extension you must request an extension using e:vision.
Requests for extension to assignment deadlines should normally be submitted before the submission deadline and may be granted for a maximum of seven days (one calendar week).

Retrieval of Failure
A pass of 40% or above must be obtained overall for the module (but not necessarily in each assessment task).
Where a student fails a module they have the right to attempt the failed assessment(s) once, at the next resit opportunity (normally July resit period). If a student fails assessment for a second time they have a right to repeat (i.e. RETAKE) the module.

NOTE: STUDENTS WHO DO NOT TAKE THEIR RESIT AT THE NEXT AVAILABLE RESIT OPPORTUNITY WILL BE REQUIRED TO REPEAT THE MODULE.

Mitigating Circumstances (also called Extenuating Circumstances).
If you are unable to meet a deadline or attend an examination, and you have a valid reason, then you will need to request via e:vision **Extenuating Circumstances.**

Feedback of assignments
You will be given feedback when you demonstrate your work.

You normally have **two working weeks** from the date you receive your grade and feedback to contact and discuss the matter with your lecturer. See the Student's Union advice page http://www.wolvesunion.org/adviceandsupport/ for more details.

Registration
Please ensure that you are registered on the module. You can check your module registrations via e:Vision You should see your personal tutor or the Student Support Officer if you are unsure about your programme of study. The fact that you are attending module classes does not mean that you are necessarily registered. A grade may not be given if you are not registered.

Cheating
Cheating is any attempt to gain unfair advantage by dishonest means and includes **plagiarism** and **collusion.** Cheating is a serious offence. You are advised to check the nature of each assessment. You must work individually unless it is a group assessment.

Cheating is defined as any attempt by a candidate to gain unfair advantage in an assessment by dishonest means, and includes e.g. all breaches of examination room rules, impersonating another candidate, falsifying data, and obtaining an examination paper in advance of its authorised release.

Plagiarism is defined as incorporating a significant amount of un-attributed direct quotation from, or un-attributed substantial paraphrasing of, the work of another.

Collusion occurs when two or more students collaborate to produce a piece of work to be submitted (in whole or part) for assessment and the work is presented as the work of one student alone.