CITS3402/CITS5507: Assignment 1

Semester 2, 2025

Assessed, worth **15%**. **Due 11:59 pm Friday 12th September**.

1 Outline

Your assignment is to produce a C library for fast parallel discrete convolutions, and to produce a report on its performance. **The project can be done individually or in a group of two.**

While this assignment is centred around programming, your submission is heavily weighted towards the report. Do not expect to receive full marks for an implementation without thorough analysis.

2 Description

Convolution is a mathematical operation that is used in a wide variety of domains, including signal processing, computer vision, machine learning, and physics/engineering. The *discrete convolution* of f and g on a finite interval $[-M, M]$ is given by,

$$(f * g)[n] = \sum_{m=-M}^{M} f[n+m]g[m]. \tag{1}$$

A widespread use of 2D convolutions is convolutional neural networks (CNNs), often used for image classification in computer vision. During inference with state-of-the-art CNNs, hundreds of thousands of convolutions are performed on high-resolution images. As a result, speeding up the convolution is very important.

$$\underbrace{\begin{pmatrix} 0.1 & 1.0 & 0.1 & 0.7 & 0.3 & 1.0 \\ 0.7 & 0.8 & 0.1 & 0.5 & 0.7 & 1.0 \\ 1.0 & 0.5 & 0.2 & 0.3 & 0.6 & 0.9 \\ 0.2 & 0.0 & 0.9 & 0.1 & 0.5 & 0.8 \\ 0.1 & 0.7 & 0.9 & 0.0 & 0.2 & 0.6 \\ 0.9 & 0.2 & 0.6 & 0.1 & 0.7 & 0.4 \end{pmatrix}}_{f} * \underbrace{\begin{pmatrix} 1.0 & 0.7 & 0.6 \\ 0.0 & 0.4 & 0.1 \\ 0.1 & 0.1 & 0.5 \end{pmatrix}}_{g} = \underbrace{\begin{pmatrix} 1.4 & 1.8 & 1.4 & 2.4 \\ 2.0 & 1.4 & 1.4 & 2.4 \\ 2.1 & 1.4 & 1.0 & 1.9 \\ 1.5 & 1.2 & 1.7 & 1.4 \end{pmatrix}}_{f * g}$$

Figure 1: Simple example of 2D convolution operation without padding.

Your task for Assignment 1 is to produce a fast parallel implementation of 2D convolution using OpenMP. See Figure 2 for an example of a 2D convolution on a 6×6 input (f) with a 3×3 kernel (g). In addition to this example, several other examples have been uploaded to the LMS as text files.

In many practical tasks, padding is often introduced to the boundaries of the input to ensure the size of the input and output are the same (often called "same" padding). With this kind of padding, zeros are added to the boundaries of the input (depending on the size of the input and kernel). For our assignment, we will always use "same" padding.

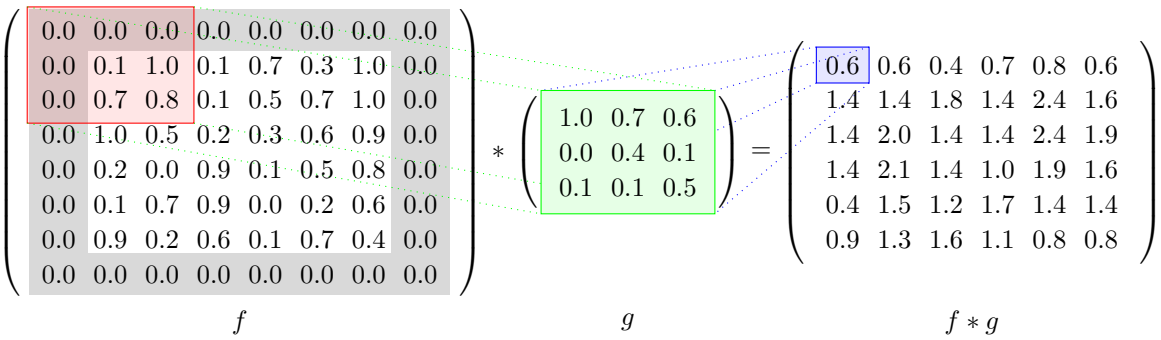

Figure 2: Example of 2D convolution operation with "same" padding.

3 Tasks

1. Implement both a single-threaded and parallel version of 2D convolution, including padding. Your implementation must work on single-precision floating-points (float32s), and be able to handle arbitrarily large arrays. Your functions should have the following basic definition:

```
void conv2d(
    float **f, // input feature map
    int H, // input height,
    int W, // input width
    float **g, // input kernel
    int kH, // kernel height
    int kW, // kernel width
    float **output
);
```

2. Implement functions for creating and reading/writing inputs (f), kernels (g), and outputs $f * g$. Your submission should include a main file that can:
   - Take two input files, and optionally write the output to a file (see Example 7.2.1).
   - Generate a random input of size $H \times W$, and kernel of size $kH \times kW$. Optionally write the input/output to a file (see Example 7.2.2).

   You may wish to use getopt to do this. Don't forget to document how to run your code in your report.

3. Ensure that your implementation produces correct results. A number of examples have been uploaded to the LMS so you can test your code. NOTE: Passing these test cases does not mean your solution is correct. You need to do thorough testing on your own.

4. Time the performance of your code. Do not include I/O (or random matrix generation) in the timing of your implementation.

5. Write a report describing your parallel implementation and its performance. Your code should include (at least) the following points:
   - Description of how you have parallelised your algorithm.
   - Description of how you are representing your arrays (f, g, $f * g$) in memory.
   - Whether you have considered the cache when developing your algorithm.
   - Thorough performance metrics and analysis of your solution describing the benefits and speedup of parallelism.

4 HPC resources

While you should do basic development and debugging locally, we expect you to use Kaya for analysing the execution time of your solution for your report.

You should stress test your program, and see what the largest inputs you can handle in under one-hour of time on a single node on Kaya. You don't have to go out of your way and maximise the size of the array, presenting the nearest power of 10 and speed-up is sufficient.

5 Submission

You should submit your assignment via LMS. The submission must include your source code (including a Makefile), and your report as a PDF in a **zip/tar file**. Please also include a **README** on how to run your code. All files must have the names and student numbers of both group members.

If we cannot build your project by simply running `make` on Kaya, you may receive zero for correctness.

6 Marking Rubric

Criteria	Proficient (100%)	Competent (66%)	Novice (33%)	Marks
Implementing serial convolution	Correct implementation. Code was not messy, well documented, and ran successfully.	Correct implementation, but code was hard to follow.	Implementation not correct with major issues.	2
Matrix generation and I/O, main file	Correct implementation fit to specifications. Code was not messy, well documented, and ran successfully.	Correct implementation fit to specifications, but code was hard to follow.	Code required minor modifications to run, or did not follow specifications correctly.	2
Implementing parallel convolution	Correct implementation with excellent use of OpenMP for parallelism.	Correct implementation with basic OpenMP parallelism.	Implementation contained minor mistakes or did not make effective use of OpenMP.	3
Description of parallelism	Thorough analysis with excellent detail.	Good analysis, but lacking minor details.	Brief analysis, significantly lacking details.	5
Description of memory layout and use of cache	Thorough analysis with excellent detail.	Good analysis, but lacking minor details.	Brief analysis, significantly lacking details.	2
Description of use of cache	Thorough analysis with excellent detail.	Good analysis, but lacking minor details.	Brief analysis, significantly lacking details.	3
Performance metrics and analysis	Thorough analysis with excellent detail.	Good analysis, but lacking minor details.	Brief analysis, significantly lacking details.	10
Formatting and general presentation	Well presented, easy to follow.	Okay presentation, but minor issues.	Poorly presented, hard to read or follow.	3
Total				30

7 Examples

7.1 File specification

An array can be stored in a space-separated text file, where the first row specifies the height and the width of the array. For example the array,

$$\begin{bmatrix} 0.884 & 0.915 & 0.259 & 0.937 \\ 0.189 & 0.448 & 0.337 & 0.033 \\ 0.122 & 0.169 & 0.316 & 0.111 \end{bmatrix} \tag{2}$$

would be represented via the text file:

```
3 4
0.884 0.915 0.259 0.937
0.189 0.448 0.337 0.033
0.122 0.169 0.316 0.111
```

This specification should be followed throughout the project.

7.2 Program output examples

7.2.1 Existing input files

An example of two existing input files producing an output:

```
$ cat f.txt
4 4
0.889 0.364 0.073 0.536
0.507 0.886 0.843 0.360
0.103 0.280 0.713 0.827
0.663 0.131 0.508 0.830

$ cat g.txt
3 3
0.485 0.529 0.737
0.638 0.168 0.338
0.894 0.182 0.314

$ ./conv_test -f f.txt -g g.txt # do not write output to file
$ ./conv_test -f f.txt -g g.txt -o o.txt # write output to o.txt
$ cat o.txt
4 4
0.643 1.532 1.484 0.956
1.230 1.802 2.079 1.705
1.195 2.466 2.189 1.798
0.417 1.340 1.572 1.247
```

7.2.2 Generating arrays

Generating arrays, without saving any outputs:

```
$ ./conv_test -H 1000 -W 1000 -kH 3 -kW 3
```

Generating arrays, saving all input and output:

```
$ ./conv_test -H 1000 -W 1000 -kH 3 -kW 3 -f f.txt -g g.txt -o o.txt
```