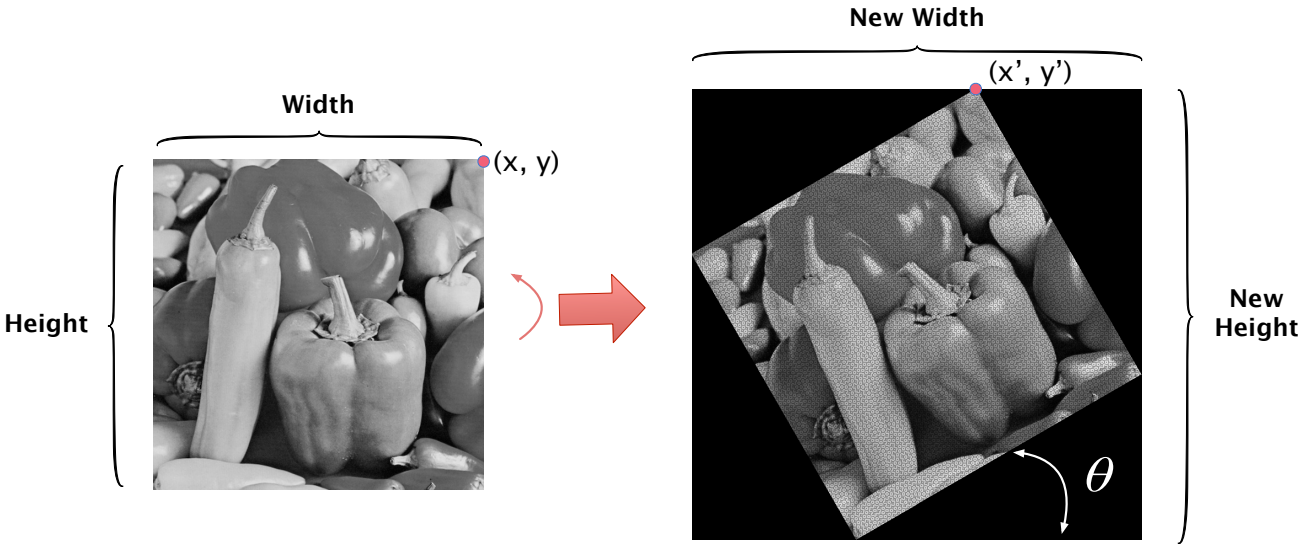
Tasks



Algorithm 1

$new\_height \leftarrow | height * cosine(\theta) | + | width * sine(\theta) |$
$new\_width \leftarrow | width * cosine(\theta) | + | height * sine(\theta) |$ ▷ Define the height and width of the new image rotated

$ori\_centre_w \leftarrow width/2$
$ori\_centre\_h \leftarrow height/2$
$new\_centre\_w \leftarrow new\_width/2$
$new\_centre\_h \leftarrow new\_height/2$
for $i = 0, 1, 2, \ldots, height - 1$ **do**
    for $j = 0, 1, 2, \ldots, width - 1$ **do**
        $x \leftarrow width - ori\_centre\_w - j$
        $y \leftarrow height - ori\_centre\_h - i$ ▷ coordinates of pixel with respect to the centre of original image

        $new\_x \leftarrow x * cosine(\theta) + y * sine(\theta)$
        $new\_y \leftarrow y * cosine(\theta) - x * sine(\theta)$
        $new\_x \leftarrow new\_centre\_w - new\_x - 1$
        $new\_y \leftarrow new\_centre\_h - new\_y - 1$ ▷ coordinates of pixel with respect to the centre of new image
        $img\_out[new\_y][new\_x] \leftarrow img\_in[i][j]$ ▷ make sure new_x and new_y are integers, and don't forget to check the boundaries
    end for
end for

0.1 Image Rotation

Image rotation is one of the most common image processing algorithms. To complete the rotation we need to write the pixel value for each pixel in the new location. Using some High School geometry

we can work out the relationship between the source coordinates (x, y) and the destination coordinates (x', y')

$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta \tag{1}$$

A complete pseudocode has been provide in Algorithm 1.

You will see some dots after this simple rotation algorithm, don't worry about that for this coursework.

0.2 Reading and Writing Image

You will need to read a PGM image, process the rotation, then write the rotated PGM image back to the file system. **Plain PGM** format is a simple grayscale graphic image format, each pixel is represented by its grey value number, with 0 being black and Maxval (defined in the PGM file) being white. The bellow image.pgm is given as an example, more detail pgm specifications can be found at http://davis.lbl.gov/Manuals/NETPBM/doc/pgm.html

image.ppm

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

1 Parallel implementations (25 points for each implementation)

Each student should implement two difference parallel programs, using MPI and one other parallel technique (Pthread, OpenMP and CUDA). Note that you only need to speedup the core algorithm.

2 Performance analysis (30 points)

Analyse the performances of two parallel implementations, including speedup and efficiencies (**15 points**).
You should provide line plots to represent the results, including serial runtime, runtime of different processes, speedup and efficiencies. (**5 points**).
Compare and contrast the two implementations (Performance Metrics, Hardware limitation, Ease of Debugging and Testing, Communication and Synchronization, or any other aspects you can think of), choose your preferred parallel technique and justify your choice (**10 points**).

3 Reflection (10 points)

Did you face any challenges during the implementation? How did you solve them?

4 Submission

You should submit the following files in a zip:

- ***rotate_mpi/pthread/openmp.c*** Your parallel implementations.
- A ***Makefile*** that will compile your code, make sure the output executable names are correct.
- A ***pdf*** file contains all the source code and the report, you should put the ***Cover Page*** in the first page of the report.

Once you have all the files, please put them in a single directory (named A2) and compress it to a ***zip*** file. You must follow the following structure:

```
|—— A2.zip
|     |—— report.pdf
|     |—— code
|     |     |—— mpi
|     |     |     |—— rotate_mpi.c
|     |     |     |—— Makefile
|     |     |—— openmp
|     |     |     |—— rotate_openmp.c
|     |     |     |—— Makefile
|     |     |—— pthread
|     |     |     |—— rotate_pthread.c
|     |     |     |—— Makefile
|     |     |—— cuda
|     |     |     |—— rotate_cuda.cu
|     |     |     |—— Makefile
```

The assignment must be submitted via Learning Mall to the correct drop box. Only electronic submission is accepted and no hard copy submission. All students must download their file and check that it is viewable after submission. Documents may become corrupted during the uploading process (e.g. due to slow internet connections). However, students themselves are responsible for submitting a functional and correct file for assessments.

Please note that quality of report and correctness of submission will also be marked (**10 points**, 5 points for the quality of report, 5 points for the correctness of submission).

Category	Exemplary (8-10)	Accomplished (6-7)	Capable (4-5)	Need Improvement (0-3)
Functionality (Correctness and output)	Code produces correct output. Demonstrate effective use of programming features as well as functions and libraries. Code is optimized for the maximum performance in parallel. The implementation follows best practices	Code produces partially correct output. Partially missing code or minor issues with excellent logic. Performance is not optimal.	Code doesn't compile. However, design idea is clear. Shows some understanding of the tasks and uses relevant programming features	Code doesn't compile. Design idea is not obvious. Doesn't know much understanding of the task.
Quality (Structure and best practice)	Excellent code quality with correct use of function and programming logic. Excellent design and the codes are commented, or design idea is clearly explained.	Code compiles. Result is faulty. Design and structure are not intuitive. Inappropriate use of function or algorithms.	Code doesn't compile. Incomplete code. However, design and structure are intuitive.	Code doesn't compile. Incomplete code. Design and structure are incorrect.
Justifications	Demonstrated correct and comprehensive justifications and analysis.	Correct justifications and analysis of algorithm with minor defects.	Limited justifications with critical errors.	Limited attempt or no justifications
Performance analysis	Provide a comprehensive analysis of performance, including detailed calculation of speedup and efficacies. Provide pleasing visualizations of results. Justifications of preferred parallel methods aligned with the actual implementations.	Provide a moderate performance analysis with mostly correct calculations. Sufficient justifications with minor mistakes.	Incomplete analysis of performance, lack of support justifications or contains major defects.	Limited attempt without any justifications.
Reflection	Reflection is thoughtful, carefully written, and demonstrates significant depth of self-reflection on the topic.	Reflection is carefully written and generally demonstrates considerable depth of self-reflection on the topic.	Reflection shows lapses in care and depth of self-reflection regarding the task at hand.	Limited attempt without any justifications.