School of Computing, University of Leeds

COMP3221
Parallel computation
Coursework 1: Shared memory parallelism with OpenMP

Deadline: 10am, Monday 28th February

If you have any queries about this coursework, visit the Microsoft Teams page for this module. If your query is not resolved by previous answers, post a new message.

This piece of work is worth **15%** of the final module grade.

Learning objectives

- Identification and handling of data dependencies in loop parallelism.
- Implementation of thread safety on a shared memory system.

Problem

Someone is trying to write an image processing application in C, and wants to develop efficient parallel implementations for various operations using OpenMP. To help with testing, they have written code that takes the input image filename as a command line argument, followed by a number corresponding to the operation to be performed. The code loads the image from file, processes the image as per the requested option, and then saves the output to file.

The image files are in uncompressed `.pgm` *(portable greymap)* format, in which each pixel takes a single integer value corresponding to its greyscale value, with 0 for black and some value `maxValue` for white. The image dimensions `width` and `height` and `maxValue` are provided in the first few lines of the image file[1].

The 4 operations to be implemented are:

1. Change light pixels to dark and *vice versa* by subtracting each pixel's greyscale value from `maxValue`. This 'negative' image is then saved as a new file `negative.pgm`.

   Function name: `saveNegativeImage()`

2. Create a mirror image in which the order of pixels within each row is reversed, so the image is flipped left–to–right. More precisely, for row `row` and column `col`, the corresponding pixel value `pixels[row][col]` should be replaced with `pixels[row][width-col-1]`. The resulting image is saved as `mirror.pgm`.

   Function name: `saveMirrorImage()`

[1] You can convert an image to `.pgm` format using ImageMagick: `convert -depth 8 -compress none <source image> <output name>.pgm`.

3. Blurs the image and outputs as `blurred.pgm`. The blurring is to be performed by sweeping through the image 10 times. For each sweep, the value of each pixel is replaced by the average of the 4 pixels surrounding that pixel, *i.e.*

   ```
   pixels[x][y]=(pixels[x-1][y]+pixels[x+1][y]+pixels[x][y-1]+pixels[x][y+1])/4;
   ```

   Pixels at the edges of the image are not changed. Your parallel algorithm should follow the red–black Gauss–Seidel pattern discussed at the end of Lecture 5.

   Function name: `saveBlurredImage()`

4. Generates a histogram of greyscale values; that is, an array that counts how many pixels have greyscale value 0, how many have value 1, *etc.* up until `maxValue`. This is then saved to a file `histogram.dat` to be displayed using a Python script `plotHistogram.py`.

   Function name: `generateHistogram()`

Your goal is to write parallel versions of each of these functions that use OpenMP to perform calculations in parallel.

Task

Download the following code from Minerva:

`cwk1.c`	:	The current version of the code from which you should start. This includes working serial implementations of options 1 and 4.
`cwk1_extra.h`	:	Contains the definition of the `Image` `struct` used to store the image, and functions to load the image, and output the modified images and the histogram. **Do not modify this file**; it will be replaced with a different version for assessment.
`image.pgm`	:	An example `.pgm` file for you to test your solution on.
`makefile`	:	A simple makefile (usage optional).
`plotHistogram.py`	:	Loads and displays the histogram output by Option 4 (usage optional).

Inspect the code until you understand how it works. In particular, see how the `main()` function calls one of the four functions in the table above, depending on the command line argument. That is, if a user were to call with

```
./cwk1 image.pgm 1
```

then `main()` loads the image `image.pgm` from file, stores it in a `struct Image` that is defined in `cwk1_extra.h`, then calls `saveNegativeImage` with a pointer to this structure.

It is your task to implement each of these functions as per the requirements above. It is expected that will only add code to the 4 functions highlighted above. You are free to add functions if you like, but you should not alter anything in `main()`.

Your submission should be as efficient as possible (in terms of parallel scalability) using the material covered up to and including lecture 6. You do not need to use any material from lectures 7 or later to achieve full marks in this coursework.

Marks

3 marks	:	Generation of negative image in parallel, in `saveNegativeImage()`.
4 marks	:	Generation of mirror image in parallel, in `saveMirrorImage()`.
4 marks	:	Generation of blurred image in parallel, in `saveBlurredImage()`.
4 marks	:	Generation of greyscale histogram in parallel, in `generateHistogram()`.

Total: 15 marks.

Submission

Submission is *via* Minerva only.

If you have only edited `cwk1.c`, submit only this file.

If you have added extra files, ensure they are all in a flat directory (*i.e.* do not use subdirectories) and that the makefile has been suitably amended. Submit all files including the new makefile as a single archive (`.tar.gz` or `.zip` formats **only**).

Do not modify `cwk1_extra.h`, **or copy any of the content to another file and then modify**. `cwk1_extra.h` will be replaced with a different version as part of the assessment.

All submissions will be compiled and executed on a School Linux machine similar to those in labs. **If you alter the makefile for any reason, ensure it still works on a School machine before submitting**.

Disclaimer

This is intended as an individual piece of work and, while discussion of the work is encouraged, what you submit should be entirely your own work. Code similarity tools will be used to check for collusion, and online source code sites will be checked.

Ensure you retain the receipt for your submission as you may be asked to produce it at a later date in the event of a disputed submission time.

The standard late penalty of 5% per day applies for work submitted after the deadline.

Examples of output (overpage)

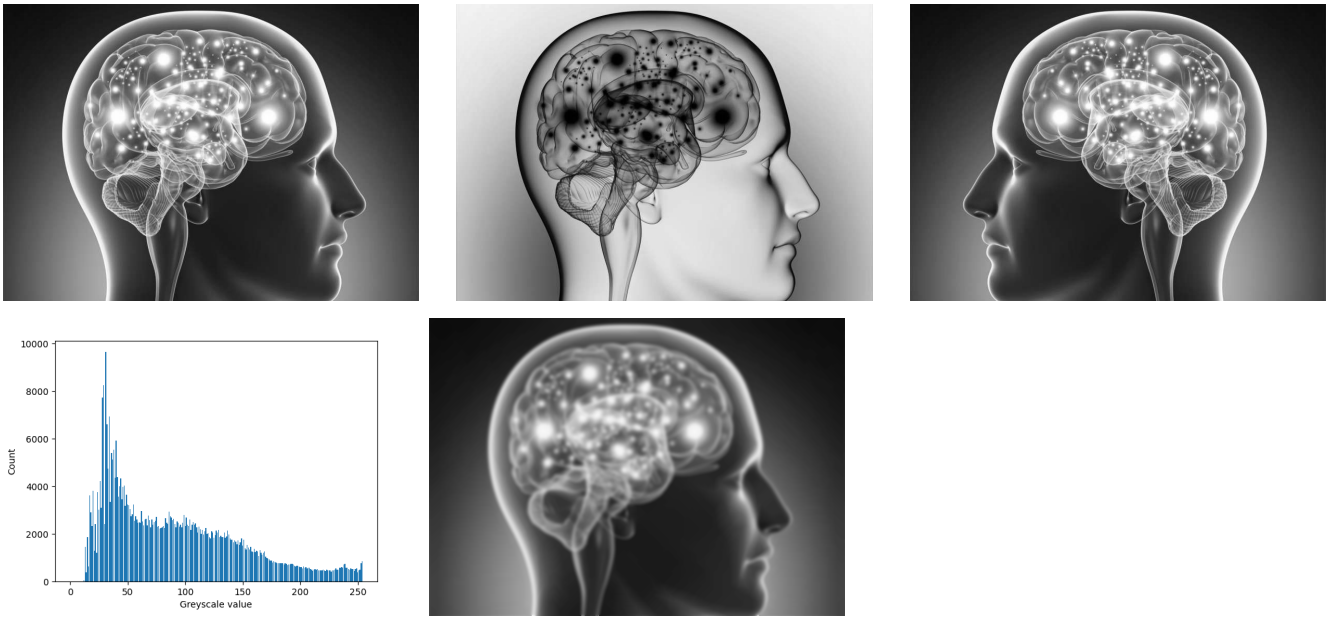

Figure 1: Clockwise from top left: Original image; negative (1); mirror image (2); blurred image (3); histogram (4).