Fundamentos de Sistemas de Operação

MIEI 2020/2021

Homework Assignment 2

“Image filtering -- *Battle of the colours*”

Deadline and Delivery

This assignment must be performed ***individually*** by each student. Any detected frauds will cause failing the discipline immediately.

The code must be submitted for evaluation via the Mooshak system using each student’s individual account (http://mooshak.di.fct.unl.pt/~mooshak/).

The deadline is ***23h59, November 20th, 2020***.

Description

Image processing is one area for which concurrent/parallel processing shows big performance gains as, commonly, individual pixels may be independently processed to produce a transformed image. The goal of this work is to parallelize an image transformation using the POSIX *Pthreads* library.

The images to be transformed follow the *RGB colour model* -- the colour of each individual pixel is defined with *three values (one byte each)* representing the *intensity* for each one of the *primary colours red*, *green*, and *blue*. In particular, the *image format* to be considered is *PPM* with *ASCII data format*. An image in this format contains

- a header that starts with the line “P3” followed by,
- a line with the number of pixels for the image’s width and height, respectively,
- a line with the maximum value for colour intensity in the colour map (e.g. 255),
- optionally, a set of comment lines that start with the cardinal character ‘#’, and
- the intensity value of a primary colour, one per line, following the *order red, green, blue*.

For example:

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
500 375
255
129
158
92
127
158
...
```

The image above has a width of 500 pixels and a height of 375 pixels, i.e. the number of lines in the image is 375 (yy dimension) and each line has 500 pixels (xx dimension). The total number of bytes necessary to represent in a vector data structure the primary colours’ intensity values for all pixels in this image is 500*375*3.

For simplification reasons, the image transformation to be considered in this work is a *point operator (pixel transform)*. Namely, to produce the new value of a pixel at position *transformed_image(i,j)*, a point operator only depends on the current value of that pixel at *original_image(i,j)*. Typical transformations are contrast adjustments, and colour correction and transformations, (e.g. negative, gray scale, darkening, etc).

The point operator to be used in this assignment *identifies the dominant primary colour*, i.e. the primary red, green, or blue that most contributes to the image, and shows the image with that primary. This colour filtering operator sums the intensity value of each one of the primary colours red, green, and blue, for all pixels in the image. The colour with the biggest value for all pixels is considered the "winner" and will remain unchanged for all pixels, whereas the "looser" colours are filtered, i.e. receive a zero intensity value. For instance, for the image example above, it is necessary to calculate

```
Sum red = 255 + 92 + …
Sum green = 129 + 127 + …
Sum blue = 158 + 158 + …
```

The biggest of these three values will identify the dominant primary colour. In case at least two colours have the same sum value, the image remains unchanged. Assuming green is the "winner" colour, the image processing has to produces a file with the following contents:

```
P3
500 375
0
129
0
0
127
0
…
```

To validate your operation transformation, you should start by producing a sequential version of your code and verify if the transformed image follows the required characteristics defined above. A bidimensional (2D) image may be represented as a vector where lines are organised sequentially:

- the size of each line is the image's width, and the image's height is the number of lines
- each pixel's colour needs three values ordered as red, green, blue
- the vector's size is *width*height*3* and may be dynamically created as

  `unsigned char image = calloc(width*height*3*sizeof(unsigned char));`
  meaning the valid entries in the vector span `image[0]` to `image[width*height*3-1]`

- at each line, the columns (yy dimension) span from *0* to *width-1*
- the line's numbers (xx dimension) span from *0* to *height-1*
- a pixel with coordinates (i,j) starts at position *(i*width+j)*3*:

```
image[(i*width+j)*3] contains the intensity of the red primary colour
image[(i*width+j)*3+1]contains the intensity of the green primary colour
image[(i*width+j)*3+2]contains the intensity of the green primary colour
```

Parallelization

To produce a version of your program with better performance, you have to parallelize your code using the pthreads POSIX library, which include syncronisation primitives such as *barrier* and *exmut* variables. The threads' launch and work division are similar to what was done during lab classes:

1. Your program creates a set of threads to divide the work so that each thread will process a part of the vector/image to calculate the sum of the colour density for each one of the primary colours for all pixels in this sub-vector
2. All threads communicate to calculate the three sums for all pixels in the image
3. All threads coordinate themselves to identify the dominant primary colour
4. All threads filter the non-dominant primary colours in their sub-vectors
5. Your program waits that all threads finish executing and write the transformed image to a file

This parallel solution is designed according to *domain decomposition* – the data in the problem domain (the image) is divided into a set of tasks (each task is the processing of a set of the image's lines), which are then statically distributed (the division is pre-defined) among the available workers (threads). Figure 1 illustrates this design.

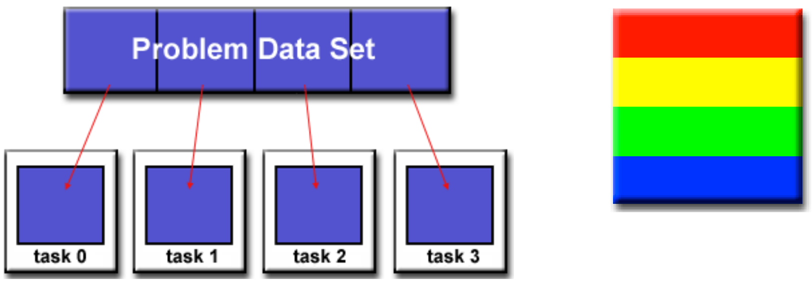

Figure 1: Domain decomposition (data parallelism) with data partition in blocks (right image) – each thread processes a set of lines [2]

The actions described above are outlined in the file `image_processor.c` in the code online.

Work to do

You are given a set of C source and header files, one Makefile, and a folder with some images in PPM format:

- `main.c` is the main program that reads an image from a file, invokes a function to process it in parallel, and writes the result to a new file.
- `image_filter_parallel.c` and `image_filter_parallel.h` define the code to create and launch the threads. The function each thread has to execute is also defined in this c file. **This is the only file you have to complete. For this, you have to fill the adequate code in the lines containing the word TODO.**
- `image_ppm.c` and `image_ppm.h` define the functions to read and write an image in PPM format, and also basic functions. These include: write the RGB intensities of a pixel; compare the colour intensities of two pixels; and filter the non-dominant primary colours. These functions have to be used in the `image_filter_parallel.c` code.
- `Makefile` defines the building rules where the generated executable is named `colour_battle`
- `images` is a folder that contains some images in the PPM format both before and after the transformation. This folder is available from the link

https://drive.google.com/drive/folders/1p5vktR8fw21jVBn3nkIY6cV4Pl9TFZUq?usp=sharing

Compile and Run the Application

To compile simply type make. The programs will compile without errors but will not work.

➢ `make`

To run you may use

➢ `./colour_battle images/image.ppm images/image_out.ppm num_threads`

Creating new PPM images

You can use the GIMP image editor to create new PPM images from jpeg, png, etc, images. For this, you have to install GIMP on your virtual machine with

- `sudo apt-get update`
- `sudo apt-get install gimp`

Open the original .jpeg, .png, etc. image with gimp, scale the image if necessary with the menu option

`Image/Scale Image (if you modify the Width, the Height will adjust accordingly)`

and generate a PPM with the menu option

`File/Export As… /Select File Type (By Extension) /PPM image /Export/Data formatting ASCII`

Submission

This homework assignment is individual and the code must be submitted for evaluation via the Mooshak system using each student's individual account (http://mooshak.di.fct.unl.pt/~mooshak/). You must submit the `image_processor.c` source file.

IMPORTANT:

- **There will be a limit of 10 submissions (per student) to mooshak.**
- **If you do less than 10 submissions, your grade is based on the last submission.**
- **If you attempt more than 10 submissions, your grade is based on the 10th submission.**

The deadline is ***23h59, November 20th, 2020***.

References

[1] Ostep – three easy pieces: http://pages.cs.wisc.edu/~remzi/OSTEP/

[2] POSIX Threads programming: https://computing.llnl.gov/tutorials/pthreads/