Universitat Pompeu Fabra Barcelona

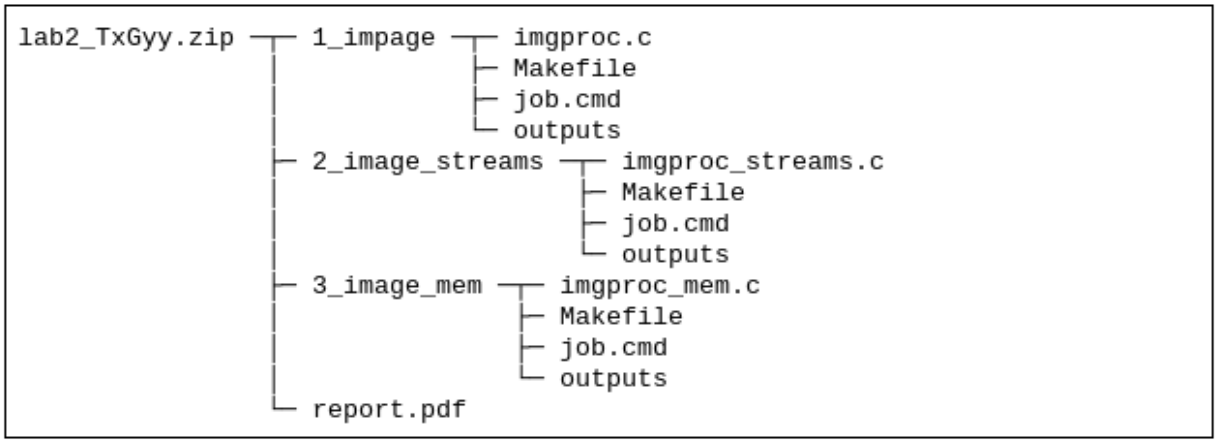
Lab 3

Ricard Borrell Pol[†], **Sergi Laut Turón**[‡]

Instructions

The third Lab session consists of 1 problem. You have to compile your answers to the exercises in a report. This lab assignment has a weight of 10% of the total grade of the subject and the deadline for submission is June 7th at 23:59.

Each group will have to submit a compressed file named lab3_TxGyy.zip, where TxGyy is your group identifier. A .tar or a .tgz file is also accepted (e.g. lab3_T1G1.zip or lab3_T2G21.zip). The compressed file has to contain three folders and all the requested files with the following structure.



A sample file named lab3_TxGyy.zip containing the reference codes has been published in the Aula Global. To ease the process we provide sample job files and Makefiles. You may need to modify the execution lines in the job scripts to perform your tests and the batch lines to request more cores or tasks. The Makefiles should not be modified unless the exercise says so. Focus on the code and the work requested for each exercise. In the case the compilation fails, the job will finish and it will not try to run the binary.

For any arising question, please, post it in the Lab class forum in the Aula Global. However, do not post your code in the Forum. For other questions regarding the assignment that you consider that cannot be posted in the Forum (e.g. personal matters or code), please, contact the responsible of the lab **sergi.laut@upf.edu**.

Criteria

The codes will be tested and evaluated on the same cluster where you will be working. The maximum grade on each part will only be given to these exercises that solve in the most specific way and that tackle all the

[†] ricard.borrell@upf.edu.
[‡] sergi.laut@upf.edu.

functionalities and work requested. All the following criteria will be applied while reviewing your labs in the cluster.

Exercises that will not be evaluated:

- A code that does not compile.
- Code giving wrong results.

Exercises with penalty:

- A code with warnings in the compilation.
- lab3_TxGyy.zip delivered files not structured as described previously.

1. Image Processing (100%)

A grayscale image can be represented as a 2D function $f(x, y)$, where:

1. x and y are the pixel coordinates,
2. $f(x, y)$ is the gray level in the range: [0,255].

We can define different transformations to the image:

- Intensity value inversion:

  $g(x, y) = 255 - f(x, y)$

- Smoothing:

  $g(x, y) = 1/9(f(x-1, y+1) + f(x, y+1) + f(x+1, y+1) + f(x-1, y) + f(x, y) + f(x+1, y) + f(x-1, y-1) + f(x, y-1) + f(x+1, y-1))$
  what is the same than applying the kernel convolution:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Edge detection:

  $g(x, y) = f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4f(x, y)$
  what it the same than applying the kernel convolution:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Edge enhancement:

  $g(x,y) = 5f(x,y) - (f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1))$
  what it the same than applying the kernel convolution:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Exercise 1 — Transformations (40%)

1 Create a code annotated with OpenACC to apply the four previously described transformations to an input grayscale. Do not take into account the pixels in the borders of the resulting images, set them to 0. Use an data region to manage GPU data allocations and transfers. Results for a reference solution with an image of a lion are included in the folder, however you can use your own image in the report. In the data provided includes:

- lleo.jpg: lion image in jpg format
- lleo.txt: lion image in txt format
- scripts/: folder with useful scripts
- sol_ref/: reference solutions for benchmarking

Exercise 2 — Asynchronous operation (30%)

1 Optimize the previous code such that the four images can be generated concurrently on the GPU by means of asynchronous operation. Moreover all the host-device and device-host data transfers must be done within the data region (so use only create and delete clauses in the data region definition). Compare the performance with previous implementation.

Exercise 3 — Quarter condition (30%)

1 Imagine that you only have capacity in the GPU for a quarter of the initial image and a quarter of the resulting image. Implement a code that applies the inversion filter in these condition. Check that you obtain the same result than with the code developed in Exercise 1.

Specifications: The parameters of the code will be the image-file-name (without extension) and the image dimensions:

```
./imgproc.o image nx ny
```

The output will be four images:

image-inverse.txt
image-smooth.txt
image-detect.txt
image-enhance.txt

as well as an screen output with the the time required to generate them, **Total time = XXs**.

Note 1: Use the python script "image_to_gray_txt.py" to convert your preferred colored image into a grayscale image saved in format txt, and the script "view_matrix.py" to visualize the images in format txt. The syntax to use these scripts is:

```
python script.py <image name without extension>
```

You can run these scripts in your own computer.

Note 2: Check that the values of the transformed image do not overpass the range [0:255].

Original

Inverse

Detect

Enhance

Smooth