Laboratory 4

(Due date: Oct. 17th)

OBJECTIVES

- Compile and execute C++ code using the TBB library in Ubuntu 12.04.4 using the Terasic DE2i-150 Development Kit.
- Perform gamma correction on a grasycale image read as a binary file.
- Execute parallel applications using TBB: *parallel_for*

TERASIC DE2I-150 DEVELOPMENT KIT

DOCUMENTATION

- Refer to the board website or the Tutorial: Embedded Intel for User Manuals and Guides.

TUTORIALS

- Refer to the Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform → *Tutorial 5* for associated examples.

ACTIVITIES

FIRST ACTIVITY: GAMMA CORRECTION APPLIED TO A GRAYSCALE IMAGE (100/100)

- Single pixel (or pixel-to-pixel) operations is a natural candidate for parallel programming as the operations are independent of each other. Among the most common operations, we can mention gamma correction, contrast enhancement, thresholding.
- In this activity, you are asked to perform gamma correction with $\gamma = 0.6$ on a grayscale image.

GAMMA CORRECTION

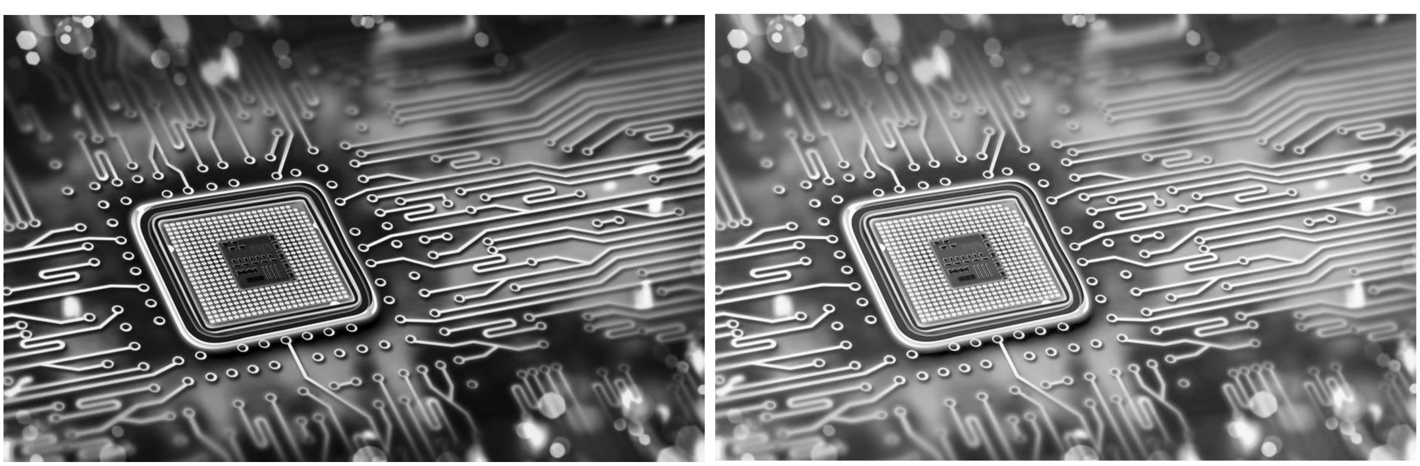
- Gamma correction is defined by $V_{out} = V_{in}{}^{\gamma}$, where $V_{out}, V_{in} \in [0,1]$. To deal with pixel values between 0 and 255, the equation needs to be re-written.

- For an 8-bit grayscale pixel IM, the resulting pixel OM is given by:

$$OM = round\left(\left(\frac{IM}{256}\right)^{\gamma} \times 256\right)$$

where IM, OM ∈ [0,255].

- Fig. 1 depicts the input grayscale image and the resulting output image (with $\gamma = 0.6$).
- Since the operations applied to every pixel are independent, this application is suitable for *TBB parallel_for*.
- To display the images, you can use the `lab4.m` script that runs in MATLAB® or GNU Octave (open-source version of MATLAB®). You can also use the script to generate input binary files from any picture you want to apply these type of pixel-to-pixel operations.



(a) (b)

Figure 1. (a) Input grayscale image. (b) Processed image with gamma correction $\gamma = 0.6$.

INSTRUCTIONS

- Write a `.cpp` program that reads a binary input file (`.bif`), computes the pixel-to-pixel operation (gamma correction with $\gamma = 0.6$), and stores the result in a binary output file (`.bof`).
  - ✓ Your code should be parallelized via *TBB parallel_for*.
  - ✓ Include a standard sequential implementation in order to compare processing times.

- **Considerations:**
  - ✓ Input matrix: Read from an input binary file (`.bif`). You can use the provided `uchip.bif` file that represents the 940x602 input image in Fig. 1(a). Each element is an unsigned 8-bit number (or `uint8`).
    - ▫ You can use the function `read_binfile` from *Laboratory 3* to read data the image data (stored as a 1D array in a raster-scan fashion) (use `typ=0` since each element is of type `uint8`).
    - ▫ You can also use the read image function available in *Tutorial #2* (for image convolution).
  - ✓ Output matrix: Elements are of type `int` (32-bit signed integer), also referred as `int32`.
    - ▫ To perform the $\left(\frac{IM}{256}\right)^{\gamma} \times 256$ operation, you need to transform the input pixel values (`uint8`) into `double` first.
      - In MATLAB®, this would be: `R = ((double(IM)/256).^0.6)*256;`
    - ▫ The $round$ operation in the equation can be achieved via rounding and saturation in C++ (this transforms a `double` type into an integer). The integer will be an `int32` type. * We could have used `uint8`, but we keep it at `int32` for simplicity's sake.
      - Rounding: For each element `x`, you can use: `if (x >= 0) xr = x + 0.5; else xr = x-0.5;`
      - Saturation: You can use typecasting: `xi = (int) xr;`
    - ▫ To store the `int32` output matrix in a `.bof` file, you can use write image code available in *Tutorial #2*.
  - ✓ *TBB - parallel_for*: <u>Be careful</u> to avoid race conditions inside the body of the *parallel_for* loop (see *Lecture Notes – Unit 4*). It is recommended to use a function to encapsulate the entire operation applied to each pixel.
  - ✓ You can use one *parallel_for* loop (after all, the input matrix is represented as a 1D array), or nested *parallel_for* loops.

- **Output matrix verification**: You need to verify the generated `.bof` file. You can do this via the `lab4.m` script. Note that when displaying, we usually saturate the matrix to [0 255]. This is performed by the script.
  - ✓ Once you place the `.bof` file in the same folder as the script, run the script. The script will display the output image generated by MATLAB as well as the output image generated by your `.cpp` code. They should match (a difference image is also displayed).

- Compile and execute the application on the DE2i-150 Board. Complete Table I (take an average over ~10 executions).

- Take a screenshot of the software running in the Terminal. It should show the computation time for both the sequential and TBB implementations
  - ✓ Your code should measure the computation time (only the actual computation portion) in us.
- Provided files: `lab4.m, uchip.jpg, uchip.bif`.

TABLE I. COMPUTATION TIME (US) OF THE GAMMA CORRECTION APPLICATION

	Computation Time (us)
Sequential	
TBB	

SUBMISSION

- Demonstration: In this Lab 4, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Webex) with the instructor and demo it (using a camera).

- Submit to Moodle (an assignment will be created):
  - ✓ One <u>.zip</u> file
    - ▫ 1st Activity: The .zip file must contain the source files (`.c, .h, Makefile`), the output binary file (`.bof`) and the requested screenshot.
  - ✓ The lab sheet (a PDF file) with the completed Table I.

TA signature: ______________________ **Date:** ______________________