

	<div style="text-align: center;">  <h1 style="margin: 0;">LAB 5 – CUDA</h1> <h2 style="margin: 10px 0 0 0;">Computação Heterogênea de Alto Desempenho (2019/2020)</h2> <p style="margin: 0;">Mestrado Integrado em Engenharia Eletrotécnica e de Computadores</p> </div>
---	---

1. Analise o seguinte kernel, `colorToGreyScaleConversion()`, que produz a conversão de cor para escala de cinzentos de uma imagem com mapeamento de dados em threads 2D:

```
// Device code
// We have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__
void colorToGreyScaleConversion(unsigned char * grayImage, unsigned char
*rgbImage, int width, int height)
{
    int Col=threadIdx.x+(blockIdx.x*blockDim.x);
    int Row=threadIdx.y+(blockIdx.y*blockDim.y);
    if (Col<width && Row<height)
    {
        // get 1D coordinate for the grayscale image
        int greyOffset=Row*width + Col;
        // one can think of the RGB image having
        // CHANNEL times columns of the gray scale image
        int rgbOffset=greyOffset*CHANNELS;
        unsigned char r=rgbImage[rgbOffset];
        // red value for pixel
        unsigned char g=rgbImage[rgbOffset+1];
        // green value for pixel
        unsigned char b=rgbImage[rgbOffset+2];
        // blue value for pixel
        // perform the rescaling and store it
        // We multiply by floating point constants
        grayImage[grayOffset]=0.21f*r + 0.71f*g + 0.07f*b;
    }
}
```

Implemente o código do host e teste-o em várias imagens, indicando o speedup obtido quando comparado com a versão sequencial (1 thread da CPU). Teste para imagens com as seguintes dimensões (width x height):

- 255x255
- 800x600
- 1920x1080

- 3840x2160
- 7680x4320

Nota: Utilize imagens à sua escolha e implemente o código de conversão da imagem em ficheiro para uma estrutura de dados do tipo unsigned char. (**Nota:** se o(a) aluno(a) perder demasiado tempo a implementar as funções de leitura/escrita da imagem, deve construir à mão a estrutura de dados que emule a imagem com informação RGB.)

2. Implemente um programa em CUDA que calcule a soma de todos os elementos de um vetor de tamanho N. Teste para vários valores de N.

- 2.1. Implemente uma versão simples (sem recorrer a optimizações).
- 2.2. Implemente uma nova versão otimizada baseada em memória partilhada.

Compare ambas as versões com a versão sequencial (1 thread da CPU).

3. Implemente um programa em CUDA que devolva a transposta de uma matriz $[A^T]_{ij} = [A]_{ji}$. Teste para vários tamanhos da matriz.

- 3.1. Implemente uma versão simples (sem recorrer a optimizações).
- 3.2. Implemente uma nova versão otimizada baseada em memória partilhada (*shared memory*).

Compare ambas as versões com a versão sequencial (1 thread da CPU).

4. Implemente um programa em CUDA que obtenha o histograma da distribuição das intensidades de uma imagem (representada em `uchar`) com N píxeis.

- 4.1. Implemente uma versão simples (sem recorrer a optimizações).
- 4.2. Implemente uma nova versão otimizada baseada em memória partilhada. Tenha em atenção que deve preservar a coerência dos dados. Para tal, investigue a utilização de operações atómicas.

5. Desenvolva um programa em CUDA que implemente a multiplicação de matrizes: $A_{m \times n} * B_{n \times p} = C_{m \times p}$.

- 5.1. Implemente uma versão simples (sem optimizações) e compare com a versão sequencial.
- 5.2. Implemente uma versão otimizada baseada em memória partilhada.

Teste para várias dimensões de matrizes e compare ambas as versões com a versão sequencial.

6. Implemente agora um programa em CUDA que aplique a operação de convolução (neste caso convolução 2D) a uma imagem. Em processamento de imagem existem vários tipos de filtros, como por exemplo *edge detection* ou *blur*, que se baseiam na operação de convolução. Para este exercício defina uma imagem e um filtro à sua escolha. Implemente duas versões:

- 6.1. Versão simples em CUDA.
- 6.2. Versão otimizada em CUDA. Nesta versão pode utilizar várias estratégias de optimização (requisito mínimo obrigatório: uso de *shared memory*).

Tenha em atenção alguns factores como:

- tamanho do filtro;
- Operações nas margens da imagem:
 - Píxeis fora da imagem replicam o valor do pixel;
 - Píxeis fora da imagem são definidos com o valor zero.

No final compare as duas versões e comente a diferença de performance.