University of Science and Technology of Hanoi
Postgraduate School

Advanced programming for HPC

LABWORK REPORT

by

LE Nguyen Khoi
M21.ICT.005

Submission Date: 23/12/2022

At the beginning of the lab work, I implement the default function of using CUDA to grayscale the image from the slide to my code file. I set the function name as grayScale_GPU. For the goal of the lab work, I also create a new function following the formula to grayscale the image by taking RGB and dividing it by 3.

In order to access to the CUDA, follow the instruction. I use @cuda.jit so that the function, so you get support for CUDA built-in variables like threadIdx

```
@cuda.jit
def grayScale_GPU(src, dst):
    tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
    g = np.uint8((src[tidx, 0] + src[tidx, 1] + src[tidx, 2]) / 3)
    dst[tidx, 0] = dst[tidx, 1] = dst[tidx, 2] = g

def grayScale_noGPU(imgArray : np.ndarray):
  for i in imgArray:
    gray = np.uint8((int(i[0])+int(i[1])+int(i[2]))/3)
    i[0] = i[1] = i[2] = gray
  imgGray = imgArray.reshape(width, height, 3)
  return imgGray
```

Before being put in the function, the image needs to be reshaped to the 1D array.

```
flatten = img.flatten().reshape(pixelCount,3)
flattenShape = np.shape(flatten)
```

To operate the function CUDA, it needs to have the number of thread-block per grid, and the number of threads per block. Because we flatten the image to the 1D array. I use the number of pixels of the image, then divide it by the number of threads per block then we have the number of thread-block per grid.

```
pixelCount = width * height
blockSize = 64
gridSize  = pixelCount / blockSize
gridSize = math.ceil(gridSize)
```

Result: The result shows a significant of difference between when using the CPU and GPU.

- Without GPU: 0.19532759600042482

- With GPU: 0.11802728800103068

When we grayscale the image, we can see it is much faster (about 0.08 seconds) for a 500x500 image when using the GPU.

Changing the block size: When changing the block size, I really don't see much difference between each in each code run times. It still gives the same result as above. So from my perspective, the number of threads per block is decided by us, depending on what situation we are using it. And it did not change the result when compiling the code.