# Report.3.cuda

Nguyen Ngoc Son 009, M2-ICT USTH 21/12/2022

## Summary

Taking the example of gray scaling image, we will compare implementation of GPU and CPU executing the same code. Taken into account looping (CPU) and parallelism (GPU), the environment of execution is under Google Colab GPU session (Tesla GPU).

Figure 1: Original RGB photo

As you can see in the figure 1, the image is RGB originally. After implementation of gray scale code with GPU, the resulting image is as below figure 2 is the same example.

Listing 1: Sample Python code – gray scale RGB image.

```
@cuda.jit
def grayscale(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
```

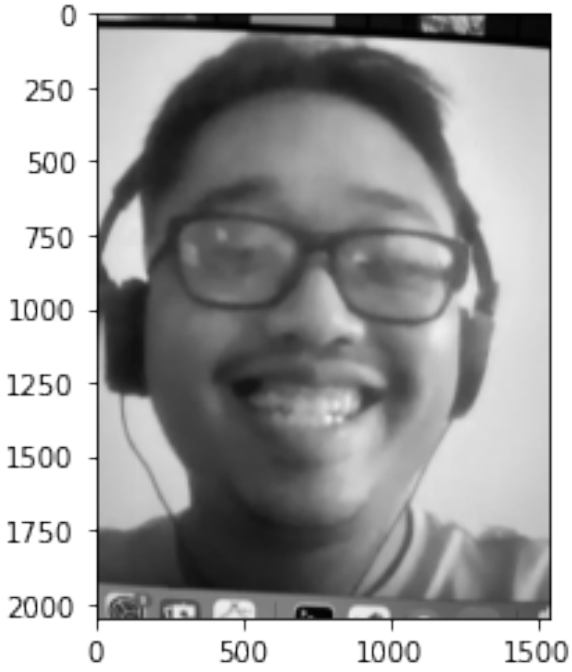


Figure 2: Gray scale photo

## CPU implementaion

Listing 2: Snip implemented on CPU

```
1 def gray(src,dst):
2     for i in range(src.shape[0]):
3       g = np.uint8((src[i][0] + src[i][1] + src[i][2])/3)
4       # print('g ', g)
5       dst[i][0] = dst[i][1] = dst[i][2] = g
```

**Output:**

Image Shape (2048, 1536, 3)

Processing time: 27.24710178375244

It took 27 seconds to processing this image on CPU. The size of the image is 1536 px in width and 2048 px in height

## GPU implementaion

Listing 3: Snip implemented on GPU

```
1
2 pixelCount = imageWidth * imageHeight
3 blockSize = 64
```

```
gridSize = int(pixelCount / blockSize)

@cuda.jit
def grayscale(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

gpu_start = time.time()
grayscale[gridSize, blockSize](devData, devOutput)
gpu_end = time.time()
```

**Output:**
Processing time: 0.1329936981201172
GPU Processing time: 0.12264299392700195

It took 0.12 second to processing this image on GPU. We can tell that with GPU implementation the processing time is very much lower. In addition this is based on block size 64. What if we execute with other block size? To answer this a graph of GPU processing time of different block size is provided below.

**block size sequence:** blockSize list = [32, 32, 64, 128, 256, 512, 1024]

Note that in this experiment, we would run block size twice (the first and second time) to see the different in time when we first load data  compile instruction at GPU. Then we expect the execution would be faster after the first time.

Listing 4: Snip implemented on GPU

```
total_processing_time =[]
gpu_processing_time = []

blockSize_list = [32, 32, 64, 128, 256, 512, 1024]

for idx, blockSize in enumerate(blockSize_list):

  gridSize = int(pixelCount / blockSize)
  image = image.copy()
  start = time.time()
  devData = cuda.to_device(image)
  devOutput = cuda.device_array(
                              (imageHeight*imageWidth, 3),
                              np.uint8)
  @cuda.jit
  def grayscale(src, dst):
```

```
17      tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
18      g = np.uint8((src[tidx, 0] + src[tidx, 1] + src[tidx, 2]) / 3)
19      dst[tidx, 0] = dst[tidx, 1] = dst[tidx, 2] = g
20
21    gpu_start = time.time()
22    grayscale[gridSize, blockSize](devData, devOutput)
23    gpu_end = time.time()
24
25    hostOutput = devOutput.copy_to_host()
26    hostOutput = hostOutput.reshape((imageHeight, imageWidth,3)).astype('↩
          uint8')
27    end = time.time()
28
29    # if idx > 0:
30    total_processing_time.append(end - start)
31    gpu_processing_time.append(gpu_end - gpu_start)
```

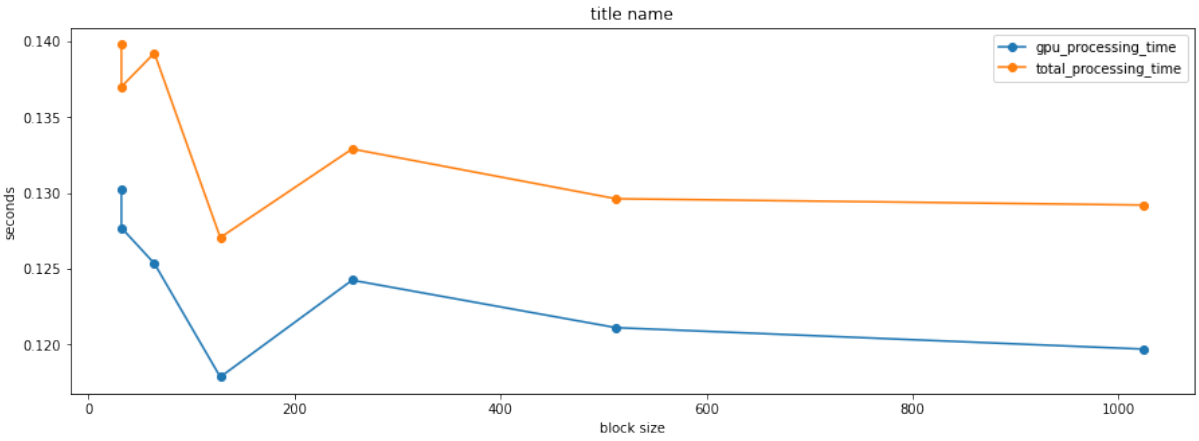


Figure 3: Result of different block size over execution time

As it is shown in the figure 3 that optimal block size is 128. The second run of is faster than the first run when instruction code is setup at GPU device.