Report.4.cuda

Nguyen Ngoc Son 009, M2-ICT USTH 24/12/2022

Summary

In this lab work, 1D and 2D GPU parallelism are compared in term of GPU processing time of grays cale. The framework used is numba in python 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 1D and 2D implementations.

```python

# 1D implementation
@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
```

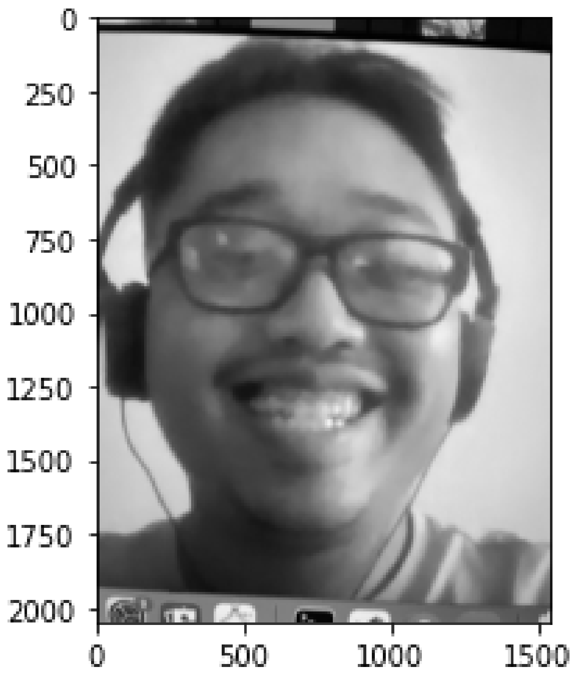

Figure 2: Gray scale photo

```

# 2D implementation
@cuda.jit
def grayscale(src, dst):
  tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
  tidy = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y
  g = np.uint8((src[tidy, tidx, 0] +src[tidy, tidx, 1] + src[tidy, ↩
      tidx, 2])/3)
  dst[tidy, tidx] = g
```

Implementations

Listing 2: 1D block size calculation

```
pixelCount = imageWidth * imageHeight
blockSize = 64
gridSize = int(pixelCount / blockSize)
```

Listing 3: 2D block size calculation

```

blockSize_list = [[1,1], [1,1], [0.5, 2], [0.5,0.5], [0.25,0.25], ↩
    [0.125,0.125]]
```

```
blockSize_list_label = [str([1*32,1*32]), str([1*32,1*32]), str↩
    ([0.5*32, 2*32]), str([0.5*32,0.5*32]), str([0.25*32,0.25*32]), str↩
    ([0.125*32,0.125*32])]

for idx, warp_ratio in enumerate(blockSize_list):
  warp_no_w = warp_ratio[0]
  warp_no_h = warp_ratio[1]
  ratio_w = int(imageWidth/(32*warp_no_w))
  ratio_h = int(imageHeight/(32*warp_no_h))

  gridSize = (ratio_w, ratio_h)
  blockSize = (int(32*warp_no_w), int(32*warp_no_h)) #max thread/block↩
      = 1024
```

warp multiplication block size sequence:

blockSize list = [[1,1], [1,1], [0.5, 2], [0.5,0.5], [0.25,0.25], [0.125,0.125]]

*Multiply these ratio with 32 (max threads/ warp) to produce the block size.

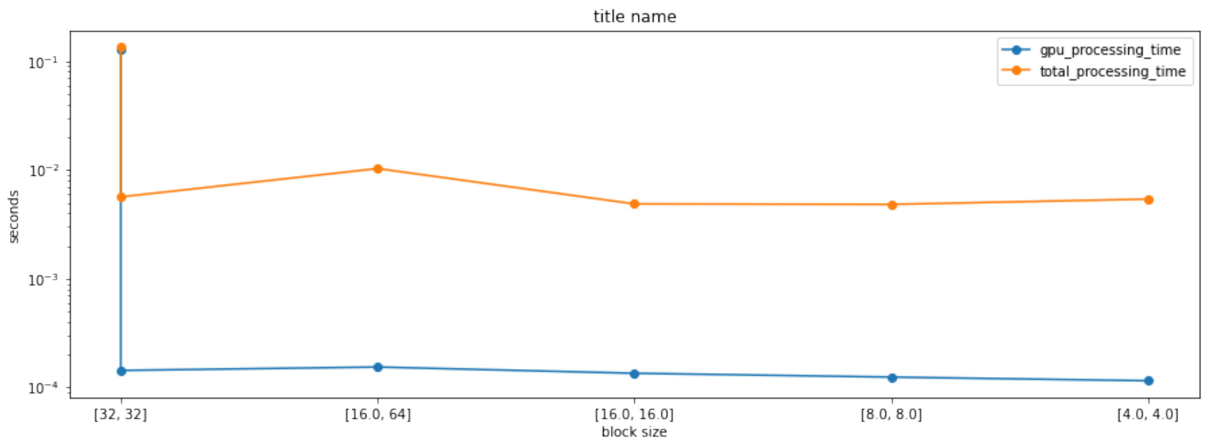

Figure 3: Result of different block size over execution time of 2D implementation

Output:

blockSize list = [[1,1], [1,1], [0.5, 2], [0.5,0.5], [0.25,0.25], [0.125,0.125]]

GPU Processing time:

"[0.128113, 0.00014, 0.00015, 0.00013, 0.00012, 0.00011]"

This is the implementation on multiple different block size with 2D array. From here we can tell block size of (4 cores, 4 cores) is fastest block size among these.

Comparing with the previous lab work running only 1D implementation, the GPU processing is less than about 1 thousands times, figure 4 !

Reference: 2D implementation code

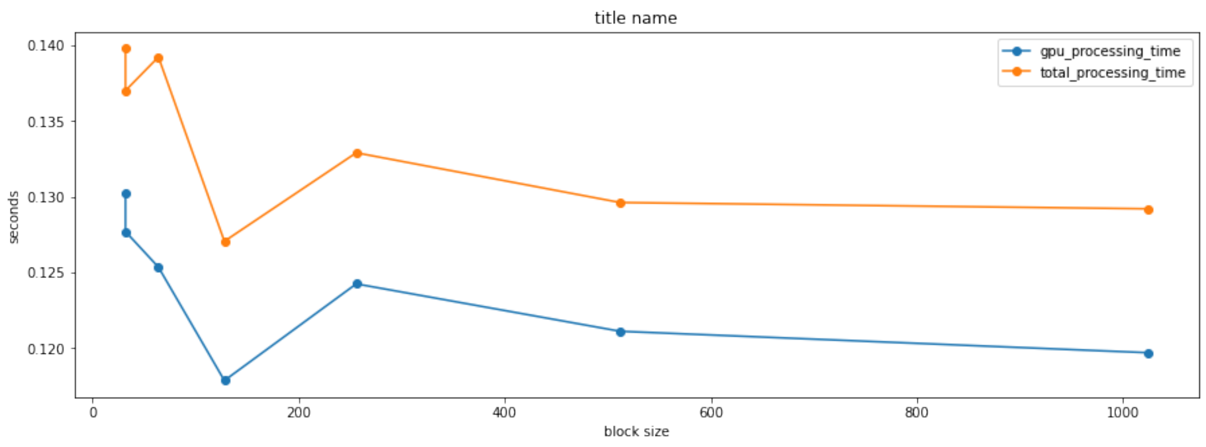

Figure 4: Result of different block size over execution time of 1D implementation

Listing 4: Snip implemented on GPU

```

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


total_processing_time =[]
gpu_processing_time = []


blockSize_list = [[1,1], [1,1], [0.5, 2], [0.5,0.5], [0.25,0.25], [0.125,0.125]]
blockSize_list_label = [str([1*32,1*32]), str([1*32,1*32]), str([0.5*32, 2*32]), str([0.5*32,0.5*32]), str([0.25*32,0.25*32]), str([0.125*32,0.125*32])]

for idx, warp_ratio in enumerate(blockSize_list):
  warp_no_w = warp_ratio[0]
  warp_no_h = warp_ratio[1]
  ratio_w = int(imageWidth/(32*warp_no_w))
  ratio_h = int(imageHeight/(32*warp_no_h))

  gridSize = (ratio_w, ratio_h)
  blockSize = (int(32*warp_no_w), int(32*warp_no_h)) #max thread/block = 1024

  image = image.copy()
  start = time.time()
```

```
devData = cuda.to_device(image)

devOutput = cuda.device_array(
                            (imageHeight, imageWidth, 3),
                            np.uint8)

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

hostOutput = devOutput.copy_to_host()
# hostOutput = hostOutput.reshape((imageHeight, imageWidth,3)).↩
   astype('uint8')
end = time.time()

# if idx > 0:
total_processing_time.append(end - start)
gpu_processing_time.append(gpu_end - gpu_start)
```