Greyscale - Labwork 3

Le Minh Hoang - 2440051

Keywords:

Abstract. No abstract for the labwork 3, I guess.

1 Introduction

This report states the work completed in Labwork 3, which will attempt to convert an image to greyscale with both CPU and GPU via CUDA.

2 Implementation

2.1 Image preprocessing

The image is flattened into a 1D array containing all RGB values sequentially

```
img = matplotlib.pyplot.imread('img.jpeg')
height, width, channels = img.shape
rgb_1d = img.reshape(height * width * 3)
```

2.2 CPU implementation

The CPU implementation uses NumPy array operations to convert the image into grayscale using the standard luminosity formula:

```
rgb_2d = rgb_1d.reshape(height, width, 3)
gray_2d = 0.299 * rgb_2d[:, :, 0] + 0.587 * rgb_2d[:, :, 1] +
    0.114 * rgb_2d[:, :, 2]
```

2.3 GPU implementation

This attempt of GPU implementation will be based on these following details

- Each CUDA thread will process one pixel (all R,G,B value)
- Thread indexing use `threadIdx.x` and `blockIdx.x` to create a unique global thread ID
- Boundary check to prevent it go out of the limit
- (R+G+B)/3 is used instead of weighted luminosity

```
@cuda.jit
def grayscale(src, dst):
    tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
    pixel_idx = tidx * 3

    if pixel_idx + 2 < src.shape[0]:
```

Article version: October 3, 2025 09:21 CET

```
        g = np.uint8((src[pixel_idx] + src[pixel_idx + 1] + src[
           pixel_idx + 2]) / 3)
        dst[pixel_idx] = g
        dst[pixel_idx + 1] = g
        dst[pixel_idx + 2] = g
```

2.4 Memory management

No idea, just copy from the slide, but I understand the concept

- Copy data from GPU –¿ CPU
- Allocate memory on GPU based on the size
- Do the calculation
- Block the CPU by waiting until the GPU finish
- Copy results from GPU to CPU

```
devSrc = cuda.to_device(rgb_1d)
devDst = cuda.device_array(height * width * 3, np.uint8)
grayscale[gridSize, blockSize](devSrc, devDst)
cuda.synchronize()
hostDst = devDst.copy_to_host()
```

2.5 Modify the Blocksize

Just the above but changes the blockSize value and plot the result

3 Results and Analysis

3.1 The CPU vs GPU

- CPU: 0.012 seconds
- GPU Init Kernel (First run): 1.3491 seconds
- GPU after kernel is ready: 0.000789 seconds

So, that's around 40x speedup

3.2 GPU CUDA with different blockSize

After changing the different blockSize variable, I achieved following result

- Block Size: 32, GPU time: 0.0003666877746582031 seconds
- Block Size: 64, GPU time: 0.0002963542938232422 seconds
- Block Size: 128, GPU time: 0.0002751350402832031 seconds
- Block Size: 256, GPU time: 0.00025725364685058594 seconds
- Block Size: 512, GPU time: 0.00025177001953125 seconds

- Block Size: 1024, GPU time: 0.0002713203430175781 seconds

That can be plotted as following graph

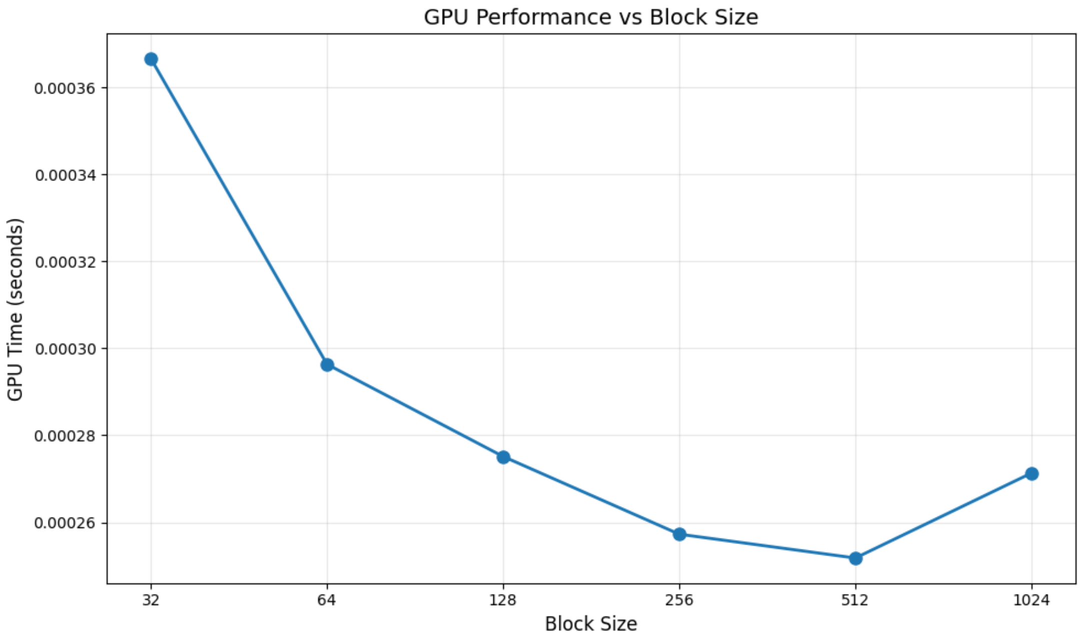

Figure 1: **GPU Algorithm Performance vs Block Size**.

3.3 Conclusion

From what have been implemented and the result that I have achieved, I can conclude that:

- The GPU is way faster than CPU when it come to the problems of multi-processing and parallel computation.
- The kernel time on the first run is huge
- Different blocksize can result in different run time, the bigger the blockSize, the better time. But it may cause to some issues related to memory when it is too big
-