CUDA Grayscale Image Conversion and Block Size Analysis

Trung Nguyen

October 7, 2025

1 Introduction

This report presents an implementation of a CUDA-based program to convert a color image into a grayscale version using parallel GPU processing. The objective is to evaluate how different CUDA block sizes affect the overall execution time of the kernel, thereby identifying the optimal configuration for performance.

2 Implementation Overview

The implementation is written in Python using the `Numba` CUDA module. The core function is a CUDA kernel named `grayscale()`, which converts each RGB pixel into a single grayscale value.

Each GPU thread is responsible for processing one pixel in the image, computing the grayscale intensity based on the average of red, green, and blue components.

```
@cuda.jit
def grayscale(src, dst):
    tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
    if tidx < src.shape[0]:
        r, g, b = src[tidx, 0], src[tidx, 1], src[tidx, 2]
        gray = (r + g + b) / 3.0
        dst[tidx, 0] = gray
        dst[tidx, 1] = gray
        dst[tidx, 2] = gray
```

Listing 1: CUDA Kernel for Grayscale Conversion

The kernel is executed with different block sizes ranging from 1 to 1024 threads per block. For each block size, the program measures the total GPU execution time and stores it for later visualization.

3 Experimental Setup

- **Input:** A color image of dimensions $(H, W, 3)$.
- **Library:** Numba CUDA.
- **Metric:** Execution time (in seconds).

- **Variable:** Block size (`threads per block`).

The grid size is computed dynamically as:

$$\text{gridSize} = \frac{ImageWidth \times ImageHeight}{\text{blockSize}}$$

where $N = H \times W$ is the total number of pixels.

4 Results and Discussion

Figure 1 shows the execution time for various block sizes. As the block size increases, the execution time typically decreases due to more efficient use of GPU resources. However, after a certain point, the performance gain may saturate or even slightly worsen due to hardware limits such as shared memory usage and scheduling overhead.

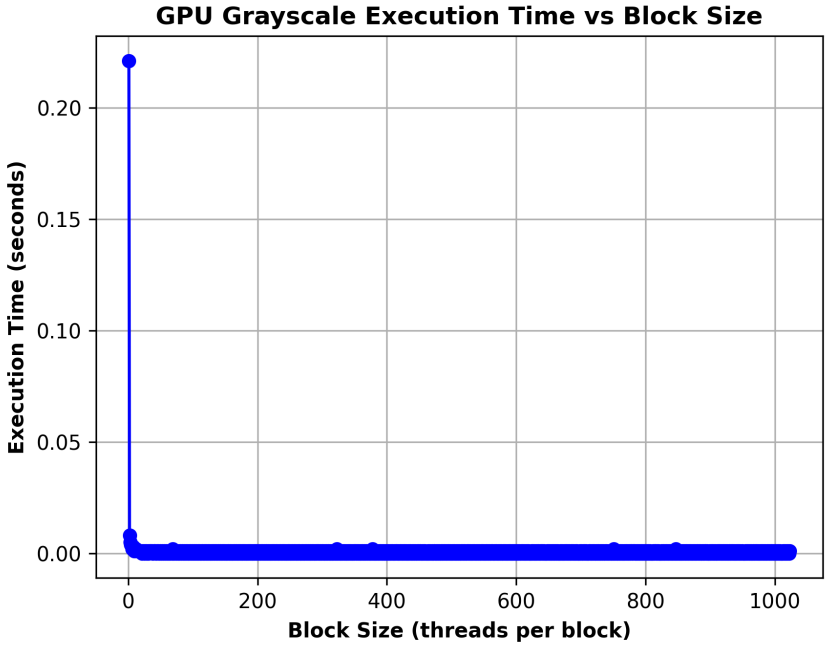

Figure 1: GPU Grayscale Execution Time vs Block Size.

The resulting grayscale image generated by the CUDA kernel is shown in Figure **??**. Each pixel's RGB values are replaced with the same intensity level, resulting in a smooth black-and-white version of the original image.

5 Conclusion

This experiment demonstrates the effectiveness of GPU-based parallelism for image processing tasks. By tuning the block size, we can significantly influence the performance of the CUDA kernel. For large images, using a higher number of threads per block often yields better execution times, although the optimal configuration depends on the specific GPU architecture.