High-Performance Computing

CUDA - Convert color image to gray image

Anand Kamble
Department of Scientific Computing
Florida State University

1 Introduction

In this assignment, we will be implementing a CUDA program to convert a color image in RGB format to a grayscale image. The implementation aims to leverage the parallel computing capabilities of CUDA to achieve better performance compared to a sequential or multi-threaded CPU implementation.

2 Implementation

In this project, we will be writing the code in a file with extension `.cu`, so that our filename will be `main.cu`. To compile this code we will be using the `nvcc` compiler which is provided by the `hpc_sdk/nvhpc-hpcx-cuda12/24.3` module. Also, we will need the `Jpegfile` library, which will be used to read the image files and write the updated image to a file.

So, the final command which we will use for compiling will be:

Listing 1: bash

```
nvcc -std=c++17 src/main.cu includes/Jpegfile.cpp -o bin/test.x
    includes/JpegLib/libjpeg.a
```

Alternatively, you can also use the `MakeFile` provided along with this project.

2.1 Host-side Initialization

The host program reads the input JPEG file using the provided `JpegFile` library, obtaining the image width, height, and pixel data in RGB format.

Listing 2: main.cu

```
// Variables to store image properties and pixel data
UINT height, width;
uint8_t *hostDataBuf;

// Read the input color image in RGB format
hostDataBuf = JpegFile::JpegFileToRGB("test-large.jpg", &width, &height);

// Allocate device memory for input and output data
uint8_t *deviceInputData, *deviceOutputData;
size_t dataSize = width * height * 3 * sizeof(uint8_t);
```

2.2 Device Memory Allocation

The host program allocates device memory for the input RGB data and output grayscale data using `cudaMalloc`.

Listing 3: main.cu

```
cudaMalloc(&deviceInputData, dataSize);
cudaMalloc(&deviceOutputData, dataSize);
```

2.3 Data Transfer to Device

The input RGB data is copied from the host memory to the device memory using `cudaMemcpy`.

Listing 4: main.cu

```
65 // Copy input data from host to device
66 cudaMemcpy(deviceInputData, hostDataBuf, dataSize, cudaMemcpyHostToDevice);
```

2.4 Kernel Launch

The convertToGrayKernel is launched on the CUDA device, with each thread responsible for converting a single pixel to grayscale. The kernel computes the grayscale value using the luminance formula:

$$gray = 0.299 * red + 0.587 * green + 0.114 * blue$$

Listing 5: main.cu

```
34 // Convert to grayscale using luminance formula
35 uint8_t gray = static_cast<uint8_t>(0.299 * red + 0.587 * green + 0.114 *
       blue);
```

3 Data Transfer to Host

After the kernel execution, the resulting grayscale data is copied from the device memory back to the host memory using `cudaMemcpy`.

Listing 6: main.cu

```
34 // Convert to grayscale using luminance formula
35 uint8_t gray = static_cast<uint8_t>(0.299 * red + 0.587 * green + 0.114 *
       blue);
```

3.1 Output File Writing

The host program writes the grayscale image data to a new JPEG file using the `JpegFile` library.

Listing 7: main.cu

```
87 // Write the grayscale image to a new JPG file
88 JpegFile::RGBToJpegFile("testmono_cuda.jpg", hostDataBuf, width, height, true,
       75);
```

3.2 Memory Deallocation

The host program frees the allocated device and host memory.

Listing 8: main.cu

```
90 // Free device memory
91 cudaFree(deviceInputData);
92 cudaFree(deviceOutputData);
93
94 // Free host memory
95 delete[] hostDataBuf;
```

4 Results

This program successfully converts color images to grayscale on execution, and we also see a major speed improvement using CUDA. With the CUDA, the program takes only 3.630720 ms to process an image of size 4000×6000.

Original Image

After Grayscale conversion

Table 1: Conversion Results

Timing compared to OpenMP and MPI: