**Extra Lab Exercise 5 – GPU vs CPU Image Processing with NPP in CUDA**

**Objective:**

To compare the performance of RGB-to-Grayscale image conversion on the CPU versus using NVIDIA Performance Primitives (NPP) on the GPU in a CUDA environment.

**Requirements:**

* NVIDIA GPU with CUDA support
* CUDA Toolkit installed (with NPP)
* C++17 compatible compiler
* BitmapPlusPlus.hpp (C++ BMP image handling library)
* BMP image file named blackbuck.bmp

**Instructions:**

**1. Setup the Project:**

* Create a folder (e.g., cuda-npp-lab)
* Place your BitmapPlusPlus.hpp and the source code (main.cu) in it
* Add blackbuck.bmp in the same directory

**Program:**

#include "BitmapPlusPlus.hpp"

#include <chrono>

#include <cuda\_runtime.h>

#include <iostream>

#include <npp.h>

#include <nppi.h>

#include <ostream>

#include <ratio>

// Error checking macro

#define CHECK\_CUDA(call) \

do { \

cudaError\_t err = call; \

if (err != cudaSuccess) { \

std::cerr << "CUDA Error: " << cudaGetErrorString(err) << std::endl; \

exit(EXIT\_FAILURE); \

} \

} while (0)

int main() {

// Image dimensions

int width = 512;

int height = 512;

int channels = 3; // RGB

bmp::Bitmap rgbImage(width, height);

rgbImage.load("blackbuck.bmp");

int imageSize = width \* height \* channels;

int graySize = width \* height;

// Allocate host memory

unsigned char \*h\_src = new unsigned char[imageSize];

unsigned char \*h\_dst = new unsigned char[graySize];

// Fill input with random RGB values

for (int y = 0; y < height; ++y) {

for (int x = 0; x < width; ++x) {

int idx = (y \* width + x) \* channels;

auto pixel = rgbImage.get( x, y); // Assumes (x, y) access returns a struct with .red/.green/.blue

h\_src[idx + 0] = pixel.r;

h\_src[idx + 1] = pixel.g;

h\_src[idx + 2] = pixel.b;

}

}

auto cpu\_start = std::chrono::high\_resolution\_clock::now();

std::vector<unsigned char> h\_gray(width \* height);

for (int y = 0; y < height; ++y) {

for (int x = 0; x < width; ++x) {

auto pixel = rgbImage.get(x, y); // Assumes (x, y) access returns a struct with .red/.green/.blue

// Convert to grayscale using the weighted sum formula

unsigned char grayscale = static\_cast<unsigned char>(0.2989 \* pixel.r + 0.5870 \* pixel.g + 0.1140 \* pixel.b);

// Store the grayscale value in the h\_gray array

h\_gray[y \* width + x] = grayscale;

}

}

auto cpu\_stop = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> cpu\_time = (cpu\_stop - cpu\_start);

// Allocate device memory

Npp8u \*d\_src = nullptr;

Npp8u \*d\_dst = nullptr;

CHECK\_CUDA(cudaMalloc((void \*\*)&d\_src, imageSize));

CHECK\_CUDA(cudaMalloc((void \*\*)&d\_dst, graySize));

// Copy input image to device

CHECK\_CUDA(cudaMemcpy(d\_src, h\_src, imageSize, cudaMemcpyHostToDevice));

// Define ROI (Region of Interest)

NppiSize oSizeROI = {width, height};

cudaEvent\_t start, stop;

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaEventRecord(start);

// Convert RGB to Grayscale

nppiRGBToGray\_8u\_C3C1R(d\_src, width \* channels, d\_dst, width, oSizeROI);

cudaEventRecord(stop);

cudaEventSynchronize(stop);

// Copy result back to host

CHECK\_CUDA(cudaMemcpy(h\_dst, d\_dst, graySize, cudaMemcpyDeviceToHost));

// Create BMP images

bmp::Bitmap gpugrayImage(width, height);

bmp::Bitmap cpuGrayImage(width, height);

// Fill gpugrayImage image with data

for (int i = 0; i < height; ++i) {

for (int j = 0; j < width; ++j) {

int idx = i \* width + j;

gpugrayImage.set(j, i, bmp::Pixel(h\_dst[idx], h\_dst[idx], h\_dst[idx]));

}

}

// Fill cpuGrayImage image with data

for (int i = 0; i < height; ++i) {

for (int j = 0; j < width; ++j) {

int idx = i \* width + j;

cpuGrayImage.set(j, i, bmp::Pixel(h\_gray[idx], h\_gray[idx], h\_gray[idx]));

}

}

// Save images

try {

// rgbImage.save("rgb\_input.bmp");

gpugrayImage.save("gpu\_grayscale\_output.bmp");

cpuGrayImage.save("cpu\_grayscale\_output.bmp");

std::cout << "Images saved: rgb\_input.bmp and grayscale\_output.bmp"

<< std::endl;

} catch (const bmp::Exception &e) {

std::cerr << "[BMP ERROR]: " << e.what() << std::endl;

}

float gpu\_time;

cudaEventElapsedTime(&gpu\_time, start, stop);

std::cout << "cpu Time taken: " << cpu\_time.count() << std::endl;

std::cout << "gpu Time taken: " << gpu\_time << std::endl;

// Cleanup

delete[] h\_src;

delete[] h\_dst;

cudaFree(d\_src);

cudaFree(d\_dst);

return 0;

}

**2. Compile the Code:**

nvcc -std=c++17 -I. rgb2gray\_npp.cu -o rgb2gray\_npp -lnppc -lnppial

* -I.: Assumes BitmapPlusPlus.hpp is in the current directory
* -lnppc, -lnppial: Link NPP Core and Image libraries

**3. Run the Executable:**

./rgb2gray\_npp

It should generate and save three BMP files:

* rgb\_input.bmp: Original image
* grayscale\_output.bmp: GPU result
* cpu\_grayscale\_output.bmp: CPU result

**Expected Output:**

Images saved: rgb\_input.bmp and grayscale\_output.bmp

cpu Time taken: <X> ms

gpu Time taken: <Y> ms