并行程序设计实践GPU项目二

项目名称：CUDA实现一个RGB向灰度图转换

完成成员名单：

班级、姓名和学号:计算机2018-5,尹浩男,201801051827

班级、姓名和学号:计算机2018-5, 乔丹 ,201801050516

班级、姓名和学号:计算机2018-5,贾志愚,201801020607

班级、姓名和学号:计算机2018-5,刘羽翯,201801120916

时间：2020年10月30日

GPU并行程序设计项目二

（1）项目要求：

用CUDA设计一个将RGB图像转换生成灰度图像的程序，要求通过实例测试串行程序和GPU并行程序的执行效率。

（2）项目源代码

#include <cuda\_runtime.h>

#include <vector>

#include <iostream>

#include <opencv2/opencv.hpp>

// 设置thread数

#define THREAD\_NUM 32

using namespace cv;

using namespace std;

// rgn图像转核函数

\_\_global\_\_ static void rgb2gray(int rows, int cols, int \*gpu\_r, int \*gpu\_g, int \*gpu\_b, int \*gpu\_gray) {

    // 得到该线程要计算的像素点的角标

    const int idx = blockIdx.x \* THREAD\_NUM + threadIdx.x;

    // 重新计算行和列

    const int row = idx / cols;

    const int column = idx % cols;

    if(row < rows && column < cols) {

        // 根据公式计算灰度值

        gpu\_gray[idx] = gpu\_r[idx] \* 0.299 + gpu\_g[idx] \* 0.587 + gpu\_b[idx] \* 0.114;

    }

}

int main() {

    // 读取源图像

    Mat src = imread("rgb.jpg");

    // 声明一个与源图像大小一致的Mat矩阵用来生成灰度图像

    Mat grayMat(src.rows, src.cols, CV\_8UC1, Scalar(0));

    // rgb三通道

    vector<Mat> rgbChannels(3);

    // r，g，b分量矩阵

    Mat rChannel, gChannel, bChannel;

    // 分离rgb

    split(src, rgbChannels);

    // 分别得到r，g，b三通道的矩阵

    rChannel = rgbChannels[0];

    gChannel = rgbChannels[1];

    bChannel = rgbChannels[2];

    // 动态申请内存空间，用来存放r，g，b的数值

    int \*r = new int[src.rows \* src.cols];

    int \*g = new int[src.rows \* src.cols];

    int \*b = new int[src.rows \* src.cols];

    int \*gray = new int[src.rows \* src.cols];

    int n = 0;

    // 分别将r，g，b三通道的矩阵赋值给动态申请的数组

    for (int i = 0; i < src.rows; ++i) {

        for (int j = 0; j < src.cols; ++j) {

            r[n] = rChannel.at<uchar>(i, j);

            g[n] = gChannel.at<uchar>(i, j);

            b[n] = bChannel.at<uchar>(i, j);

            n += 1;

        }

    }

    // 申明显存上用来存储rgb分量的数组，以及存储灰度值的数组

    int \*gpu\_r, \*gpu\_g, \*gpu\_b, \*gpu\_gray;

    // 在显存上申请空间

    cudaMalloc((void\*\*)&gpu\_r, sizeof(int) \* src.rows \* src.cols);

    cudaMalloc((void\*\*)&gpu\_g, sizeof(int) \* src.rows \* src.cols);

    cudaMalloc((void\*\*)&gpu\_b, sizeof(int) \* src.rows \* src.cols);

    cudaMalloc((void\*\*)&gpu\_gray, sizeof(int) \* src.rows \* src.cols);

    // 将内存上的rgb数组拷贝到显存上

    cudaMemcpy(gpu\_r, r, sizeof(int) \* src.rows \* src.cols, cudaMemcpyHostToDevice);

    cudaMemcpy(gpu\_g, g, sizeof(int) \* src.rows \* src.cols, cudaMemcpyHostToDevice);

    cudaMemcpy(gpu\_b, b, sizeof(int) \* src.rows \* src.cols, cudaMemcpyHostToDevice);

    // 计算耗时

    cudaEvent\_t start, stop;

    float elapsedTime;

    cudaEventCreate(&start);

    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);

    // 执行rgb图像转灰度图的核函数

    rgb2gray << <(src.rows \* src.cols + THREAD\_NUM - 1) / THREAD\_NUM, THREAD\_NUM, 0>> >(src.rows, src.cols, gpu\_r, gpu\_g, gpu\_b, gpu\_gray);

    // 结束计时

    cudaEventRecord(stop, 0);

    cudaEventSynchronize(start);

    cudaEventSynchronize(stop);

    cudaEventElapsedTime(&elapsedTime, start, stop);

    printf("GPUtime: %f ms\n", elapsedTime);

    cudaEventDestroy(start);

    cudaEventDestroy(stop);

    // 将灰度值拷贝回内存

    cudaMemcpy(gray, gpu\_gray, sizeof(int) \* src.rows \* src.cols, cudaMemcpyDeviceToHost);

    // 释放显存上动态申请的空间

    cudaFree(gpu\_r);

    cudaFree(gpu\_g);

    cudaFree(gpu\_b);

    cudaFree(gpu\_gray);

    n = 0;

    // 将灰度值赋值给灰度图矩阵用来生成灰度图像

    for (int i = 0; i < src.rows; ++i) {

        for (int j = 0; j < src.cols; ++j) {

            grayMat.at<uchar>(i, j) = gray[n];

            n += 1;

        }

    }

    Mat merge\_img;

    vector<Mat> channels\_all;

    // 整合像素值

    channels\_all.push\_back(grayMat);

    merge(channels\_all, merge\_img);

    // 生成图像

    imwrite("gray.jpg", merge\_img);

    // Opencv方法转灰度图

    // 声明矩阵存放灰度值

    Mat cpu\_gray(src.rows, src.cols, CV\_8UC1, Scalar(0));

    // 开始计时

    clock\_t cpu\_st = clock();

    // 非并行方法将rgb图像转灰度图

    for (int y = 0; y < src.rows; y++) {

              uchar\*cp = src.ptr<uchar>(y);

              uchar\*gp = cpu\_gray.ptr<uchar>(y);

              for (int x = 0; x < src.cols; x++) {

                       \*gp = (15\*cp[0] + 75\*cp[1] + 38\*cp[2]) >> 7;

                       cp += 3;

                       gp++;

              }

    }

    // 结束计时

    clock\_t cpu\_et = clock() - cpu\_st;

    printf("CPUtime: %lf ms\n", (double)(cpu\_et)/CLOCKS\_PER\_SEC \* 1000);

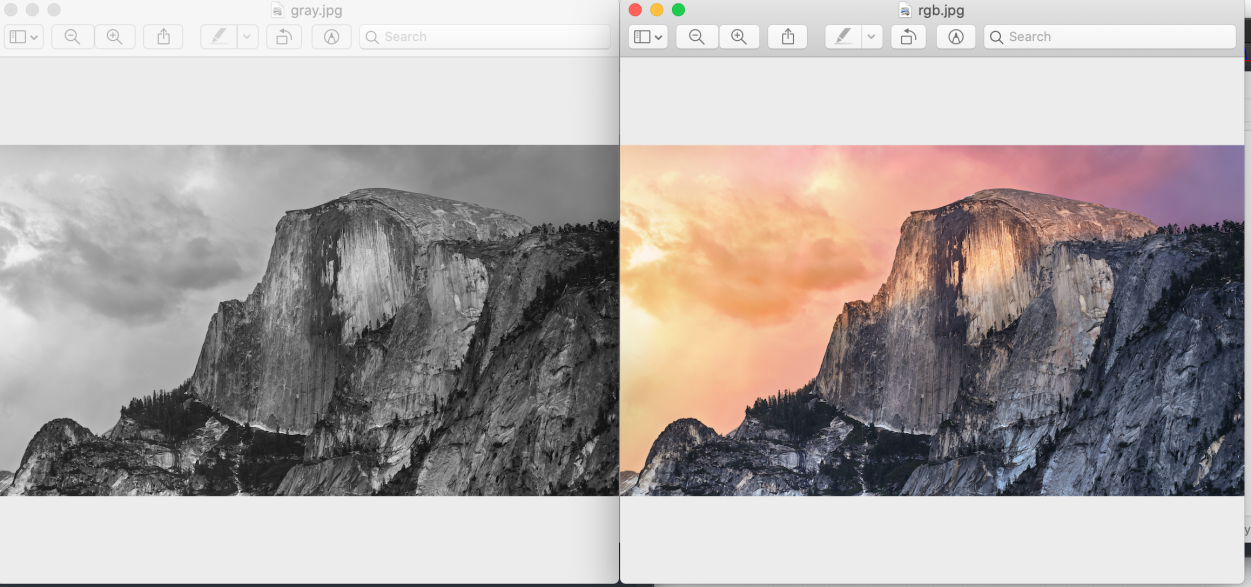
    // 写入图像

    imwrite("cpu\_gray.jpg", cpu\_gray);

    return 0;

}

（3）程序运行结果



GPU耗时: 2.360640 ms

CPU耗时: 50.000000 ms