Homework10 CUDA卷积加速测试

2101210578 胡成成

Problem

Sobel 常用于图像的边缘检测，计算公式如下：

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

其中 **A**是二维图像，**G**为检测到的梯度强度。
请用**GPU**实现，并与上次作业中的方法比较计算时间。

Answer

源代码

- Cuda代码

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <cuda.h>
#include <device_functions.h>
#include <opencv2\opencv.hpp>
#include <iostream>

#include "time.h"

using namespace std;
using namespace cv;

//Sobel using CPU
void sobel(Mat srcImg, Mat dstImg, int imgHeight, int imgWidth)
{
    float Gx = 0;
    float Gy = 0;
    for (int i = 1; i < imgHeight - 1; i++)
    {
        uchar* dataUp = srcImg.ptr<uchar>(i - 1);
        uchar* data = srcImg.ptr<uchar>(i);
        uchar* dataDown = srcImg.ptr<uchar>(i + 1);
```

```
        uchar* out = dstImg.ptr<uchar>(i);
        for (int j = 1; j < imgWidth - 1; j++)
        {
            Gx = (dataUp[j + 1] + 2 * data[j + 1] + dataDown[j + 1]) - (dataUp[j
- 1] + 2 * data[j - 1] + dataDown[j - 1]);
            Gy = (dataUp[j - 1] + 2 * dataUp[j] + dataUp[j + 1]) - (dataDown[j -
1] + 2 * dataDown[j] + dataDown[j + 1]);

            if (Gx < 0) Gx = 0;
            if (Gx > 255) Gx = 255;

            if (Gy < 0) Gy = 0;
            if (Gy > 255) Gy = 255;

            out[j] = sqrt(Gx * Gx + Gy * Gy);
        }
    }
}

//Sobel using OpenMP
void sobelOpenMP(Mat srcImg, Mat dstImg, int imgHeight, int imgWidth)
{
    float Gx = 0;
    float Gy = 0;
    #pragma omp parallel for private(Gx, Gy)
    for (int i = 1; i < imgHeight - 1; i++)
    {
        uchar* dataUp = srcImg.ptr<uchar>(i - 1);
        uchar* data = srcImg.ptr<uchar>(i);
        uchar* dataDown = srcImg.ptr<uchar>(i + 1);
        uchar* out = dstImg.ptr<uchar>(i);
        for (int j = 1; j < imgWidth - 1; j++)
        {
            Gx = (dataUp[j + 1] + 2 * data[j + 1] + dataDown[j + 1]) - (dataUp[j
- 1] + 2 * data[j - 1] + dataDown[j - 1]);
            Gy = (dataUp[j - 1] + 2 * dataUp[j] + dataUp[j + 1]) - (dataDown[j -
1] + 2 * dataDown[j] + dataDown[j + 1]);

            if (Gx < 0) Gx = 0;
            if (Gx > 255) Gx = 255;

            if (Gy < 0) Gy = 0;
            if (Gy > 255) Gy = 255;

            out[j] = sqrt(Gx * Gx + Gy * Gy);
        }
    }
}


//Sobel using Cuda
__global__ void sobelInCuda(unsigned char* dataIn, unsigned char* dataOut, int
imgHeight, int imgWidth)
{
    int xIndex = threadIdx.x + blockIdx.x * blockDim.x;
    int yIndex = threadIdx.y + blockIdx.y * blockDim.y;
    int index = yIndex * imgWidth + xIndex;
    float Gx = 0;
```

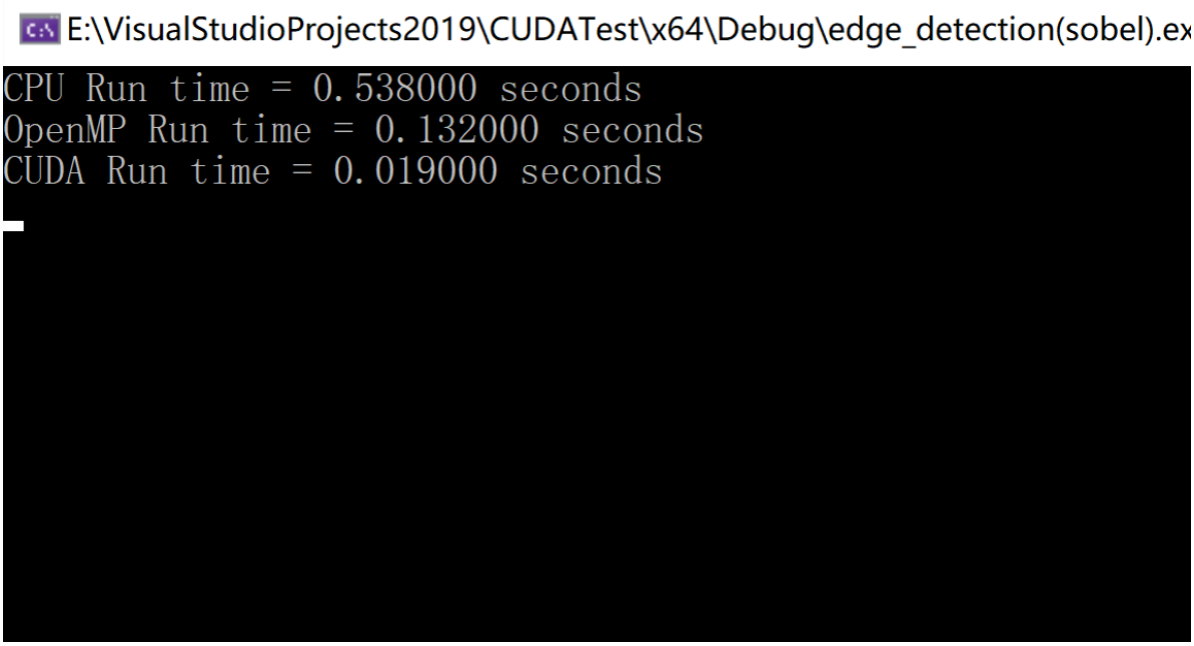
```
    float Gy = 0;

    if (xIndex > 0 && xIndex < imgWidth - 1 && yIndex > 0 && yIndex < imgHeight
- 1)
    {
        Gx = dataIn[(yIndex - 1) * imgWidth + xIndex + 1] + 2 * dataIn[yIndex *
imgWidth + xIndex + 1] + dataIn[(yIndex + 1) * imgWidth + xIndex + 1]
            - (dataIn[(yIndex - 1) * imgWidth + xIndex - 1] + 2 * dataIn[yIndex
* imgWidth + xIndex - 1] + dataIn[(yIndex + 1) * imgWidth + xIndex - 1]);
        Gy = dataIn[(yIndex - 1) * imgWidth + xIndex - 1] + 2 * dataIn[(yIndex -
1) * imgWidth + xIndex] + dataIn[(yIndex - 1) * imgWidth + xIndex + 1]
            - (dataIn[(yIndex + 1) * imgWidth + xIndex - 1] + 2 * dataIn[(yIndex
+ 1) * imgWidth + xIndex] + dataIn[(yIndex + 1) * imgWidth + xIndex + 1]);

        if (Gx < 0) Gx = 0;
        if (Gx > 255) Gx = 255;

        if (Gy < 0) Gy = 0;
        if (Gy > 255) Gy = 255;

        dataOut[index] = sqrt(Gx * Gx + Gy * Gy);
    }
}


int main()
{
    // Read RGB as gray
    Mat grayImg = imread("ship.jpg", 0);

    int imgHeight = grayImg.rows;
    int imgWidth = grayImg.cols;

    Mat gaussImg;
    // Noise reduction
    GaussianBlur(grayImg, gaussImg, Size(3, 3), 0, 0, BORDER_DEFAULT);

    //Sobel result by using CPU
    Mat dst(imgHeight, imgWidth, CV_8UC1, Scalar(0));


    clock_t cpu_start = clock();
    sobel(gaussImg, dst, imgHeight, imgWidth);
    clock_t cpu_finish = clock();
    double duration = (double)(cpu_finish - cpu_start) / CLOCKS_PER_SEC;
    printf("CPU Run time = %f seconds\n", duration);

    openmp_start = clock();
    sobelOpenMP(gaussImg, dst, imgHeight, imgWidth);
    openmp_finish = clock();
    duration = (double)(openmp_finish - openmp_start) / CLOCKS_PER_SEC;
    printf("OpenMP Run time = %f seconds\n", duration);

    //Sobel result by using CUDA
    Mat dstImg(imgHeight, imgWidth, CV_8UC1, Scalar(0));

    //Define GPU input and output pointer
    unsigned char* d_in;
```

```
    unsigned char* d_out;

    cudaMalloc((void**)&d_in, imgHeight * imgWidth * sizeof(unsigned char));
    cudaMalloc((void**)&d_out, imgHeight * imgWidth * sizeof(unsigned char));

    //Copy data from CPU to GPU
    cudaMemcpy(d_in, gaussImg.data, imgHeight * imgWidth * sizeof(unsigned
char), cudaMemcpyHostToDevice);

    dim3 threadsPerBlock(32, 32);
    dim3 blocksPerGrid((imgWidth + threadsPerBlock.x - 1) / threadsPerBlock.x,
(imgHeight + threadsPerBlock.y - 1) / threadsPerBlock.y);

    cuda_start = clock();

    sobelInCuda <<< blocksPerGrid, threadsPerBlock >>> (d_in, d_out, imgHeight,
imgWidth);

    // GPU and CPU timer Synchronize
    cudaDeviceSynchronize();
    cuda_finish = clock();

    duration = (double)(cuda_finish - cuda_start) / CLOCKS_PER_SEC;
    printf("CUDA Run time = %f seconds\n", duration);

    //Copy data from GPU to CPU
    cudaMemcpy(dstImg.data, d_out, imgHeight * imgWidth * sizeof(unsigned char),
cudaMemcpyDeviceToHost);

    //Free Cuda
    cudaFree(d_in);
    cudaFree(d_out);


    namedWindow("dst_image", WINDOW_FREERATIO);
    imshow("dst_image", dstImg);

    waitKey(0);
    destroyAllWindows();


    return 0;
}
```

- 测试原始图像

- 运行结果对比：

运算方式	运行时间（秒）
CPU	0.538000
OpenMP	0.132000
CUDA	0.019000



- Sobel算子计算后的图像

结论

- Cuda编程的速度大于OpenMP大于单线程CPU，可见GPU的运算速度相当快。

遇到问题

- CUDA中int类型不支持sqrt操作，需要变量类型为double型
- OpenCV的MAT对象无法直接参与Cuda计算，需要使用data属性存入指针参与计算
- 在运行之前确定threadsPerBlock的尺寸需要看自己GPU的属性，例如：

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include "device_functions.h"

int getThreadNum()
{
    cudaDeviceProp prop;//cudaDeviceProp的一个对象
    int count = 0;//GPU的个数
    cudaGetDeviceCount(&count);
    cout << "gpu 的个数: " << count << '\n';

    cudaGetDeviceProperties(&prop, 0);//第二参数为那个gpu
    cout << "最大线程数: " << prop.maxThreadsPerBlock << endl;
    cout << "最大网格类型: " << prop.maxGridSize[0] << '\t' << prop.maxGridSize[1]
<< '\t' << prop.maxGridSize[2] << endl;
    return prop.maxThreadsPerBlock;
}
```