CS4990 Fall 2024 Homework 3

Total points: 100

Due date: Friday, November 15, 2024

Task Description:

In this assignment, you are tasked with developing a complete CUDA C/C++ program for an image blur application, also known as image smoothing that we learned in Module 3 "Multidimensional Grids and Data".

Convolution serves as the fundamental operation for implementing the image blur process. Specially, one of the convolution kernels requested in this assignment should be optimized using the tiled convolution technique that we learned in Module 7 "Convolution". This optimization involves leveraging GPU shared memory and constant memory to enhance performance.

Below are the specific requirements:

1. **Develop** a single CUDA program file named **"convolution.cu"** containing all the necessary code to blur an input image and generate its blurred image. For simplicity, we will use the average filter of size 5×5 in this assignment, where each filter element holds the floating-point value 1/25.

   It is highly recommended to utilize the NCSA Delta GPUs for this assignment. However, if you're experienced in successfully building and installing OpenCV for C++ from sources, you may proceed with your own computers.

   Below are the command-lines for compiling and executing your program using NCSA Delta GPUs:

   - Log in to the NCSA supercomputer using your own NCSA account.
   - Enter an interactive session with GPUs, for example:
     srun --account=bchn-delta-gpu --partition=gpuA40x4-interactive --nodes=1 --gpus-per-node=1 --tasks=1 --tasks-per-node=16 --cpus-per-task=1 --mem=20g --pty bash
   - Load the OpenCV module:
     module load opencv/4.9.0.x86_64
   - To compile:
     nvcc -o convolution convolution.cu -I $OPENCV_HOME/include/opencv4/ -L $OPENCV_HOME/lib64 -lopencv_core -lopencv_imgcodecs -lopencv_imgproc
   - To execute**:**
     ./convolution inputImg.jpg

   When running your program,

   - Please make sure to replace "inputImg.jpg" with the name of your input image file, which should be located in the same directory as your program. For example, if

your input image is named "santa-grayscale.jpg", the execution command would be "./convolution santa-grayscale.jpg"
- No command argument for filter radius is necessary in this assignment, as we utilize the constant average filter of size 5×5 where the filter radius is 2.

2. Within "convolution.cu", **implement** one host function and two CUDA kernels to perform convolution, respectively. Specifically,
    a. A host function that performs the convolution operation using CPU-only.
    b. A CUDA kernel that performs the convolution operation using GPU but without tiling. You may refer to the example provided on Slide 40 in Module 3, but **please modify it to incorporate the average filter matrix**. It is acceptable to load the average filter from either GPU global memory or constant memory.
    c. **An optimized CUDA kernel** that presents **a "tiled" version of the convolution operation using GPU shared memory**. Specifically, in this optimized kernel, please **load the average filter from GPU constant memory**.

3. **Ensure** that your code can handle images with **varying image sizes**. Please also **consider** boundary conditions to ensure proper handling in such cases.

    For testing purposes, two input images of different sizes are provided in the zipped folder accompanying this assignment.
    - "santa-grayscale.jpg": a grayscale image of size $1{,}000 \times 1{,}000$
    - "tree-grayscale.jpg": a grayscale image of size 345×346

    You may also choose to test your program with additional grayscale images if desired.

4. **Utilize** timing techniques such as CPU timers or CUDA events, **to measure the performance of** your implementation of the host function and two CUDA kernels as specified above.

We also suggest **structuring** the "convolution.cu" **by implementing the following macros, host functions, and CUDA kernels**. At the end of this assignment, we will provide screenshots of an example of the program's structure.

- **#define CHECK(call)**
    - A macro for error checking.
- **double myCPUTimer()**
    - A timer for measuring execution time.
- **void blurImage_h(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)**
    - A host function for CPU-only convolution.
- **__global__ void blurImage_Kernel(unsigned char * Pout, unsigned char * Pin, unsigned int width, unsigned int height)**
    - A CUDA kernel performs a simple convolution without using tiling.
- **void void blurImage_d(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)**
    - A host function for handling device memory allocation and free, data copy, and calling the specific CUDA kernel, blurImage_Kernel().

- **__global__ void blurImage_tiled_Kernel(unsigned char * Pout, unsigned char * Pin, unsigned int width, unsigned int height)**
    - A CUDA kernel that performs a "tiled" version of convolution using GPU shared memory and loading the filter elements from GPU constant memory.
- **void blurImage_tiled_d(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)**
    - A host function for handling device memory allocation and free, data copy, and calling the specific CUDA kernel, blurImage_tiled_Kernel().
- **int main(int argc, char** argv)**
    - The main entry point of the program
- **bool verify(cv::Mat answer1, cv::Mat answer2, unsigned int nRows, unsigned int nCols)**
    - A function to validate if the blurred image result using a CUDA kernel matches that of the OpenCV's blur function, cv::blur().
    - Note that you may call this verification function **three** times in your main function, in order to
        - Compare the blurred image result of blurImage_h() with that of cv::blur().
        - Compare the blurred image result of blurImage_d() with that of cv::blur().
        - Compare the blurred image result of blurImage_tiled_d() with that of cv::blur().

Please note that if needed, you may review how to use OpenCV with CUDA programs by referring to the CUDA&&OpenCV program examples we covered during lectures on Module 3.

What to Submit?

Please **compress** the following **six** required files into a zip file, named following the format "yourname_p3.zip", and **submit** the zipped file on Canvas.

1. **"convolution.cu":** a single CUDA program file containing all required code.
2. **"blurredImg_opencv.jpg"**: the blurred image generated by OpenCV's cv::blur(), given an input image of your choice.
3. **"blurredImg_cpu.jpg"**: the blurred image generated by blurImage_h() using CPU-only, given the same input image.
4. **"blurredImg_gpu.jpg"**: the blurred image generated by blurImage_d() using GPU but without tiling, given the same input image.
5. **"blurredImg_tiled_gpu.jpg"**: the blurred image generated by blurImage_tiled_d() using GPU and the tiling, given the same input image.
6. **"output.jpg"**: a screenshot presenting the verification results of the "verify" function and the timing results.

To demonstrate the necessary information for your screenshot, here's an example of a screenshot displaying the output of my vector-addition program. It includes the verification results and the timing results. However, please note that this example is not for convolution.

```
[vccjihao@gpub001 CS4990]$ ./main
Vector size 16777216
vecAdd on CPU:                                  0.035908 s

    cudaMalloc:                                 0.000372 s
    cudaMemcpy:                                 0.010065 s
    vecAddKernel<<<(32768,1,1),(512,1,1)>>>:    0.025956 s
    cudaMemcpy:                                 0.009967 s
vecAdd on GPU:                                  0.048771 s

Verifying results...TEST PASSED
```

Below please find screenshots of an example for the program's structure for this assignment. Please note that the CUDA program shown in the screenshots is incomplete. You also need to modify the part of the program responsible for loading an input image specified by the user. For example,

- Parse the command-line arguments to extract the filename of the input image specified by the user.
- Load the specified input image dynamically at runtime, instead of using a hardcoded filename.
- Ensure error handling for cases where the specified image file cannot be found or loaded properly.

```
#include <opencv2/opencv.hpp>
#include <sys/time.h>

#define FILTER_RADIUS 2

// for simplicity, we use the constant average filter only in this assignment
const float F_h[2*FILTER_RADIUS+1][2*FILTER_RADIUS+1] = {
    {1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25},
    {1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25},
    {1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25},
    {1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25},
    {1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25, 1.0f / 25}
};

__constant__ float F_d[2*FILTER_RADIUS+1][2*FILTER_RADIUS+1];

// check CUDA error if exists
#define CHECK(call){ \
    const cudaError_t cuda_ret = call; \
    if(cuda_ret != cudaSuccess){ \
        printf("Error: %s:%d, ", __FILE__, __LINE__); \
        printf("code: %d, reason:%s\n", cuda_ret, cudaGetErrorString(cuda_ret)); \
        exit(-1); \
    } \
}

// check if the difference of two cv::Mat images is small
bool verify(cv::Mat answer1, cv::Mat answer2, unsigned int nRows, unsigned int nCols) {
  const float relativeTolerance = 1e-2;
  for(int i=0; i<nRows; i++){
        for(int j=0; j<nCols; j++){
            float relativeError = ((float)answer1.at<unsigned char>(i,j) - (float)answer2.at<unsigned char>(i,j))/255;
            if (relativeError > relativeTolerance || relativeError < -relativeTolerance) {
                printf("TEST FAILED at (%d, %d) with relativeError: %f\n", i, j, relativeError);
                printf("    answer1.at<unsigned char>(%d, %d): %u\n", i, j, answer1.at<unsigned char>(i,j));
                printf("    answer2.at<unsigned char>(%d, %d): %u\n\n", i, j, answer2.at<unsigned char>(i,j));
                return false;
            }
        }
  }
  printf("TEST PASSED\n\n");
  return true;
}
```

```
44
45  // CPU timer
46  double myCPUTimer(){
47      struct timeval tp;
48      gettimeofday(&tp, NULL);
49      return ( (double)tp.tv_sec + (double)tp.tv_usec/1.0e6);
50  }
51
52  // A CPU-implementation of image blur using the average box filter
53  void blurImage_h(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)
54  {
55
56  }
57
58  // A CUDA kernel of image blur using the avergage box filter
59  __global__ void blurImage_Kernel(unsigned char * Pout, unsigned char * Pin, unsigned int width, unsigned int height)
60  {
61
62  }
63
64  // A GPU-implementation of image blur using the average box filter
65  void blurImage_d(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)
66  {
67
68  }
69
70  // An optimized CUDA kernel of image blur using the avergage box filter from constant memory
71  __global__ void blurImage_tiled_Kernel(unsigned char * Pout, unsigned char * Pin, unsigned int width, unsigned int height)
72  {
73
74  }
75
76  // A GPU-implementation of image blur, where the kernel performs shared memory tiled convolution using the average box filter from constant memory
77  void blurImage_tiled_d(cv::Mat Pout_Mat_h, cv::Mat Pin_Mat_h, unsigned int nRows, unsigned int nCols)
78  {
79
80  }
81
82  int main(int argc, char** argv){
83      cudaDeviceSynchronize();
84
85      double startTime, endTime;
86
87      // use OpenCV to load a grayscale image.
88      cv::Mat grayImg = cv::imread("santa-grayscale.jpg", cv::IMREAD_GRAYSCALE);
89      if(grayImg.empty()) return -1;
90
91      // obtain image's height, width, and number of channels
92      unsigned int nRows = grayImg.rows, nCols = grayImg.cols, nChannels = grayImg.channels();
93
94      // for comparsion purpose, here uses OpenCV's blur function which uses average filter in convolution
95      cv::Mat blurredImg_opencv(nRows, nCols, CV_8UC1, cv::Scalar(0));
96      startTime = myCPUTimer();
97      cv::blur( grayImg, blurredImg_opencv, cv::Size( 2*FILTER_RADIUS+1, 2*FILTER_RADIUS+1 ), cv::Point(-1, -1), cv::BORDER_CONSTANT);
98      endTime = myCPUTimer();
99      printf("openCV's blur (CPU):                        %f s\n\n", endTime - startTime); fflush(stdout);
100
101     // for comparsion purpose, implement a cpu version
102     cv::Mat blurredImg_cpu(nRows, nCols, CV_8UC1, cv::Scalar(0));   // cv:Mat constructor to create and initialize an cv:Mat object; note that CV_8UC1 implies 8-bit
    unsigned, single channel
103     startTime = myCPUTimer();
104     blurImage_h(blurredImg_cpu, grayImg, nRows, nCols);
105     endTime = myCPUTimer();
106     printf("blurImage on CPU:                                %f s\n\n", endTime - startTime); fflush(stdout);
107
108     // implement a gpu version that calls a CUDA kernel
109     cv::Mat blurredImg_gpu(nRows, nCols, CV_8UC1, cv::Scalar(0));
110     startTime = myCPUTimer();
111     blurImage_d(blurredImg_gpu, grayImg, nRows, nCols);
112     endTime = myCPUTimer();
113     printf("blurImage on GPU:                                %f s\n\n", endTime - startTime); fflush(stdout);
114
115     // implement a gpu version that calls a CUDA kernel which performs a shared-memory tiled convolution, and filter elements are loaded from constant memory
116     cv::Mat blurredImg_tiled_gpu(nRows, nCols, CV_8UC1, cv::Scalar(0));
117     startTime = myCPUTimer();
118     blurImage_tiled_d(blurredImg_tiled_gpu, grayImg, nRows, nCols);
119     endTime = myCPUTimer();
120     printf("(tiled)blurImage on GPU:                            %f s\n\n", endTime - startTime); fflush(stdout);
121
122     // save the result blurred images to disk
123     bool check = cv::imwrite("./blurredImg_opencv.jpg", blurredImg_opencv);
124     if(check == false){ printf("error!\n"); return -1;}
125
126     check = cv::imwrite("./blurredImg_cpu.jpg", blurredImg_cpu);
127     if(check == false){ printf("error!\n"); return -1;}
128
129     check = cv::imwrite("./blurredImg_gpu.jpg", blurredImg_gpu);
130     if(check == false){ printf("error!\n"); return -1;}
131
132     check = cv::imwrite("./blurredImg_tiled_gpu.jpg", blurredImg_tiled_gpu);
133     if(check == false){ printf("error!\n"); return -1;}
134
135     // check if the result blurred images are similar to that of OpenCV's
136     verify(blurredImg_opencv, blurredImg_cpu, nRows, nCols);
137     verify(blurredImg_opencv, blurredImg_gpu, nRows, nCols);
138     verify(blurredImg_opencv, blurredImg_tiled_gpu, nRows, nCols);
139
140     return 0;
141 }
```