Assignment № 1

JI Zhuoran, The University of Hong Kong 23/08/2018

Most of the GPUs are Single Instruction Multiple Data(SIMD) execution model. By the very nature all thread performs same instruction in parallel, at least conceptually. If we want to perform different operations on the different thread in same block, say, if A, then op1, else op2, then we essentially force these thread to do op1 first and then op2. This "unneccessary" serialization carries performance penalty. The aim of this tutorial are:

- test those cost
- try to avoid this penalty

Problem 1

For the first problem in your first assignment, let's write an interesting program.

You need to write a code to draw slices of the Julia Set, as shown in figure **??**. For the uninitiated, the Julia Set is the boundary of a certain class of functions over complex numbers. Undoubtedly, this sounds not that interesting. However, for almost all values of the function's parameters, this boundary forms a fractal, one of the most interesting and beautiful curiosities of mathematics.

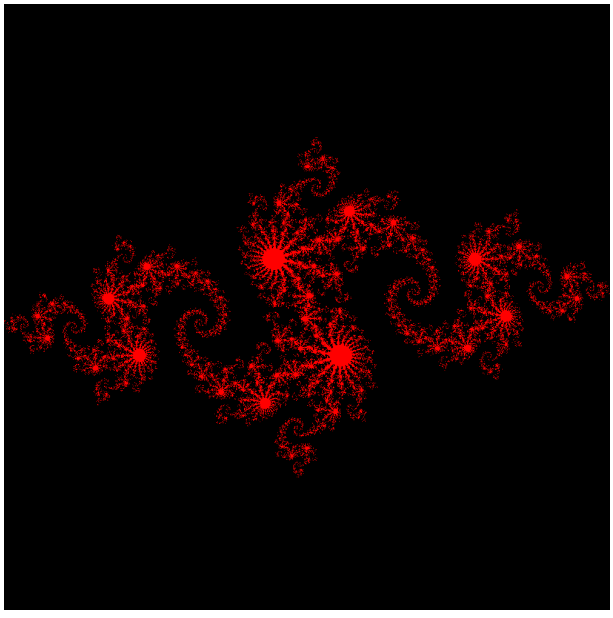

Figure 1: The figure of Julia Set you are going to generate.

The calculations involved in generating such a set are quite simple. At its heart, the Julia Set evaluates a simple iterative equation for points in the complex plane. A point is not in the set if the process of iterating the equation diverges for that point. That is, if the sequence of values produced by iterating the equation grows toward infinity, a point is considered outside

the set. Conversely, if the values taken by the equation remain bounded, the point is in the set. Computationally, the iterative equation in question is remarkably simple: $Z_{n+1} = Z_n^2 + C$. Computing an iteration of this equation would therefore involve squaring the current value and adding a constant to get the next value of the equation.

As this is not a math course, we have already implemented a CPU version Julia Set Generator in **julia.cu**. All you need to do is rewriting the CPU code to a GPU one, and hopefully optimize the code to beat otehrs.

- generate the Julia Set using GPU correctly. (55%)
- write a tiny report (one paragraph is enough) to compare the execution time between CPU and GPU. (5%)
- code style, error check, memory free etc. (10%)
- write a tiny report to explain the optimization method you use and compare the execution time between them. HINT GIVEN IN TUTORIAL(30%)

Problem 2

In this problem, you are going to blur an image (a poster of iron man, if you do not like him, feel free to use others, as long as it is large enough and in ppm format). To do this, imagine that we have a filter, which is a square array of weight values. For each pixel in the image, imagine that we overlay this square array of weights on top of the image such that the center of the weight array is aligned with the current pixel. To compute a blurred pixel value, we multiply each pair of numbers that line up. In other words, we multiply each weight with the pixel underneath it. Finally, we add up all of the multiplied numbers and assign that value to our output for the current pixel. We repeat this process for all the pixels in the image.

Noticed that the input is a color image that has three channels. The image is coded as RGBARGBARGBARGBA..., the first elements is the red value of the first pixel, and the second elements is the green value of the first pixel, so on so forth. You need to blur the three channel separately.

You must fill in the **your_gaussian_blur** function in **blur.cu** to perform the blurring of the **PPMImage**, using the array of weights as below, and put the result in the **PPMImage**. After the **your_gaussian_blur** function, you need to save the image for grading.

Here is an example of computing a blur, using a weighted average, for a single pixel in a small image.

Array of weights:

0.05	0.1	0.05
0.1	0.4	0.1
0.05	0.1	0.05

Image:

1	2	5	2	0	3
3	2	5	1	6	0
4	3	6	2	1	4
0	4	0	3	4	2
9	6	5	0	3	9

The red 6 will be replaced by $2 * 0.05 + 5 * 0.1 + 6 * 0.05 + 3 * 0.1 + 6 * 0.4 + 2 * 0, 1 + 4 * 0.05 + 0 * 0.1 + 3 * 0.05$.

- generate the blur image using GPU correctly. (45%)
- write a tiny report (one paragraph is enough) to compare the execution time between CPU and GPU. (5%)
- code style, error check, memory free etc. (10%)
- write a tiny report to explain the optimization method you use and compare the execution time between them. HINT GIVEN IN TUTORIAL (40%)

Reference

DOCUMENT PROVIDED BY NVIDIA(For detail please send email to comp3231.hku@gmail.com)