Programming Assignment V: CUDA Programming

Parallel Programming by Prof. Yi-Ping You

Due date: **23:59, Dec 16, Thursday, 2021**

The purpose of this assignment is to familiarize yourself with CUDA programming.

Table of Contents

Get the source code:

```
$ wget https://nycu-sslab.github.io/PP-f21/HW5/HW5.zip
$ unzip HW5.zip -d HW5
$ cd HW5
```

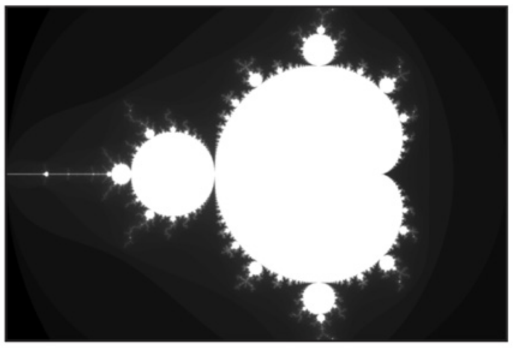
1. Problem Statement: Paralleling Fractal Generation with CUDA

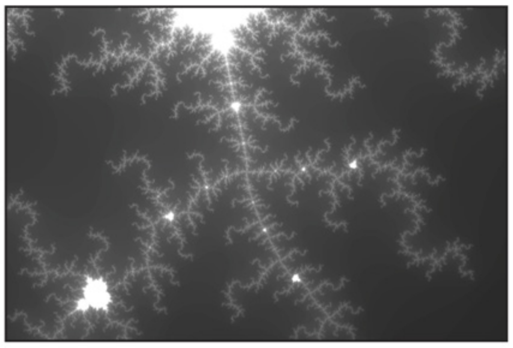
Following part 2 of HW2, we are going to parallelize fractal generation by using CUDA.

Build and run the code in the `HW5` directory of the code base. (Type `make` to build, and `./mandelbrot` to run it. `./mandelbrot --help` displays the usage information.)

The following paragraphs are quoted from part 2 of HW2.

> This program produces the image file `mandelbrot-test.ppm`, which is a visualization of a famous set of complex numbers called the Mandelbrot set. [Most platforms have a `.ppm` viewer. For example, to view the resulting images, use tiv command (already installed) to display them on the terminal.]
>
> As you can see in the images below, the result is a familiar and beautiful fractal. Each pixel in the image corresponds to a value in the complex plane, and the brightness of each pixel is proportional to the computational cost of determining whether the value is contained in the Mandelbrot set. To get image 2, use the command option `--view 2`. You can learn more about the definition of the Mandelbrot set.


View 1


View 2
(66x zoom)

Your job is to parallelize the computation of the images using CUDA. A starter code that spawns CUDA threads is provided in function `hostFE()`, which is located in `kernel.cu`. This function is the host front-end function that allocates the memory and launches a GPU kernel.

Currently `hostFE()` does not do any computation and returns immediately. You should add code to `hostFE()` function and finish `mandelKernel()` to accomplish this task.

The kernel will be implemented, of course, based on `mandel()` in `mandelbrotSerial.cpp`, which is shown below. You may want to customized it for your kernel implementation.

```
int mandel(float c_re, float c_im, int maxIteration)
{
  float z_re = c_re, z_im = c_im;
  int i;
  for (i = 0; i < maxIteration; ++i)
  {

    if (z_re * z_re + z_im * z_im > 4.f)
      break;

    float new_re = z_re * z_re - z_im * z_im;
    float new_im = 2.f * z_re * z_im;
    z_re = c_re + new_re;
    z_im = c_im + new_im;
  }

  return i;
}
```

2. Requirements

1. You will modify only `kernel.cu`, and use it as the template.
2. You need to implement three approaches to solve the questions:
    1. Method 1: Each CUDA thread processes one pixel. Use `malloc` to allocate the host memory, and use `cudaMalloc` to allocate GPU memory. Name the file `kernel1.cu`. (*Note that you are not allowed to use the image input as the host memory directly*)
    2. Method 2: Each CUDA thread processes one pixel. Use `cudaHostAlloc` to allocate the host memory, and use `cudaMallocPitch` to allocate GPU memory. Name the file `kernel2.cu`.
    3. Method 3: Each CUDA thread processes a group of pixels. Use `cudaHostAlloc` to allocate the host memory, and use `cudaMallocPitch` to allocate GPU memory. You can try different size of the group. Name the file `kernel3.cu`.
3. **Q1** What are the pros and cons of the three methods? Give an assumption about their performances.
4. **Q2** How are the performances of the three methods? Plot a chart to show the differences among the three methods
    - for VIEW 1 and VIEW 2, and
    - for different `maxIteration` (1000, 10000, and 100000).

You may want to measure the running time via the `nvprof` command to get a comprehensive view of performance.

1. **Q3** Explain the performance differences thoroughly based on your experimental results. Does the results match your assumption? Why or why not.
2. **Q4** Can we do even better? Think a better approach and explain it. Implement your method in `kernel4.cu`.

Answer the questions (marked with "**Q1-Q4**") in a **REPORT** using HackMD. Notice that in this assignment a higher standard will be applied when grading the quality of your report.

3. Grading Policy

NO CHEATING!! You will receive no credit if you are found cheating.

Total of 100%:

- Implementation correctness: 54%
    - `kernel1.cu`: 18%
    - `kernel2.cu`: 18%
    - `kernel3.cu`: 18%
    - For each kernel implementation,
        - (8%) the output (in terms of both views) should be correct for any `maxIteration` between 256 and 100000, and
        - (10%) the speedup over the reference implementation should always be greater than 0.6x (kernel1 & kernel2) / 0.3x (kernel3) for `maxIteration` between 256 and 100000 regarding VIEW 1.
- Competition: 20%
    - Use `kernel4.cu` to compete with your classmates. The competition is judged by the running times of your program for generating both views for `maxIteration` of 100000 with the metric below.
        - VIEW 1: 10%
        - VIEW 2: 10%
- Questions: 26%
    - Q1: 5%
    - Q2: 5%
    - Q3: 10%
    - Q4: 6%

Metric for each view:

$$\frac{T-Y}{T-F} \times 60\%, \text{if } Y < T + \begin{cases} 20\%, \text{if } Y < F \times 2 \\ 40\%, \text{if } Y < F \times 1.5 \end{cases}$$

where Y and F indicate the execution time of your program and the fastest program, respectively, and $T = F \times 1.5$.

4. Evaluation Platform

Your program should be able to run on UNIX-like OS platforms. We will evaluate your programs on the workstations dedicated for this course. You can access these workstations by `ssh` with the following information.

The workstations are based on Ubuntu 20.04 with Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz processors and GTX 1060 6GB. `g++-10`, `clang++-11`, and `cuda10.1` have been installed.

IP	Port	User Name	Password
140.113.215.195	37072 ~ 37080	{student_id}	{your_passwd}

Login example:

```
$ ssh <student_id>@140.113.215.195 -p <port>
```

You can use the testing script `test_hw5` to check your answer *for reference only*. Run `test_hw5` in a directory that contains your `HW5_XXXXXXX.zip` file on the workstation.

5. Submission

All your files should be organized in the following hierarchy and zipped into a `.zip` file, named `HW5_xxxxxxx.zip`, where `xxxxxxx` is your student ID.

Directory structure inside the zipped file:

- `HW5_xxxxxxx.zip` (root)
  - `kernel1.cu`
  - `kernel2.cu`
  - `kernel3.cu`
  - `kernel4.cu`
  - `url.txt`

Notice that you just need to provide the URL of your HackMD report in `url.txt`, and enable the write permission for someone who knows the URL so that TAs can give you feedback directly in your report.

Zip the file:

```
$ zip HW5_xxxxxxx.zip kernel1.cu kernel2.cu kernel3.cu kernel4.cu url.txt
```

Be sure to upload your zipped file to new E3 e-Campus system by the due date.

You will get *NO POINT* if your ZIP's name is wrong or the ZIP hierarchy is incorrect.

6. References

1. CUDA C++ Programming Guide
2. cudaMallocPitch API Document
3. CUDA 開發環境設定與簡易程式範例
4. CPU 與 GPU 計算浮點數的差異