CMDA 3634 Fall 2017 Homework 05

Kevin Jiang

December 6, 2017

You must complete the following task by 11:59pm on 11/28/17.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Click on this link
2. Click on Menu/Copy Project.
3. Modify the HW05.tex document to respond to the following questions.
4. Remember: click the Recompile button to rebuild the document when you have made edits.
5. Remember: Change the author
6. Instructions for assignment referred to in **Q1** available on Canvas.

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

1. `firstnameLastnameHW05.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW05.tex` file.
3. `firstnameLastnameHW05.pdf` PDF file.
4. `cudaMandelbrot.cu` and `cudaJulia.cu` text file with student code.

You must complete this assignment on your own.

90 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

Q1 *(30 points) CUDA Mandelbrot.*

30 points will be awarded for completing the "adding CUDA to a Mandelbrot set generator" class assignment. Follow the instructions in the assignment PDF closely (available on Canvas). Remember to push the code to a HW5 sub directory in a repository that Dr. Warburton and William Winter have access to. Include the URL for the repository in your assignment write up.

Copy and paste the contents of the batch script file used to submit a job to newriver into your LaTeX report. Include the plot mentioned in (k) of the instructions in your report. Submit code to canvas (as well as pushing code to your GitHub repository).

```
#! /bin/bash

#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=1:gpus=1
#PBS -W group_list=newriver
#PBS -q p100_dev_q
#PBS -A CMDA3634

cd $PBS_O_WORKDIR

module purge
module load cuda

nvcc -O3 -o mandelbrot -arch=sm_60 mandelbrot.cu png_util.c -I. -lpng -lm

./mandelbrot 4096 4096 32
./mandelbrot 4096 4096 64
./mandelbrot 4096 4096 128
./mandelbrot 4096 4096 256
./mandelbrot 4096 4096 512

rm mandelbrot

echo "Exited script normally"
```

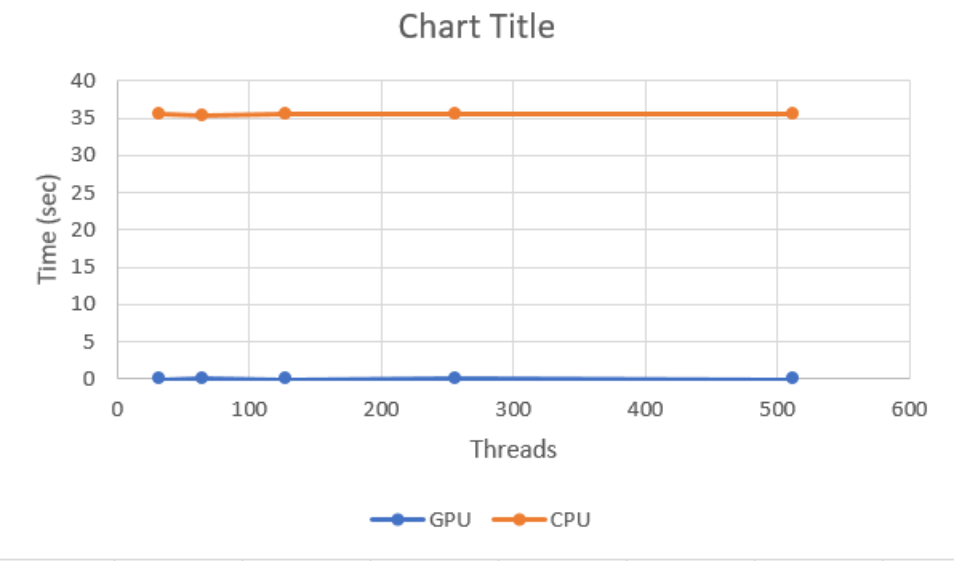

Figure 1: GPU vs CPU rendering times

Q2 *(30 points) CUDA Filled Julia Set.*

Now we adapt the CUDA code from **Q1** to compute the Filled Julia Set for a particular value of c. When computing the Mandelbrot set, we fix an initial starting point $z = 0$ and vary c. To compute the Julia Set, we fix a particular c and vary the starting point z. Copy the `cudaMandelbrot.cu` code completed in **Q1** to a new file called `cudaJulia.cu`. Edit the methods of the `cudaJulia.cu` file according to the following instructions.

First edit the main function:

1. Set `centRe` to 0, `centIm` to 0 and diam to 1.2.
2. Change `cmin`, `cmax`, and `dc` to `zmin`, `zmax`, and `dz` respectively.
3. Create a `complex_t` type variable called c. Set c's imaginary part to 0.1560 and c's real part to -0.8.
4. Have the program call your `julia` kernel instead of `mandelbrot`. `julia` will have to take c as an additional input. (See the instructions for the `julia` kernel below.)
5. Change all instances of `mandelbrot` to `julia` in the remainder of the method.

Next change the `mandelbrot` kernel to a `julia` kernel:

1. Choose the starting value for the complex variable z based on the two-dimensional thread and block indices. (Remember: We use a different z value to initialize the iteration for each pixel of the final Julia image for a *globally fixed* c.)
2. Add an additional `complex_t` input that is the fixed c used in the iteration.
3. Have threads call `testpoint`. (See below for changes to be made to `testpoint`.)
4. Store the result of the `testpoint` method in the `julia` array.

Lastly, update the `testpoint` method:

1. Change the prototype so `testpoint` takes two `complex_t` variables (z and c).
2. Since z and c are both inputs, delete the `complex_t` z declaration from the method.

Include the image generated by the program with the specified value of c in your report ($c = -0.8 + 0.156i$). Change values of c, `centRe`, `centIm`, and `diam` to produce different images. Extra credit will be awarded for cool Julia images. Submit code to canvas (as well as pushing code to your GitHub repository).

c value: -0.8 + 0.156i

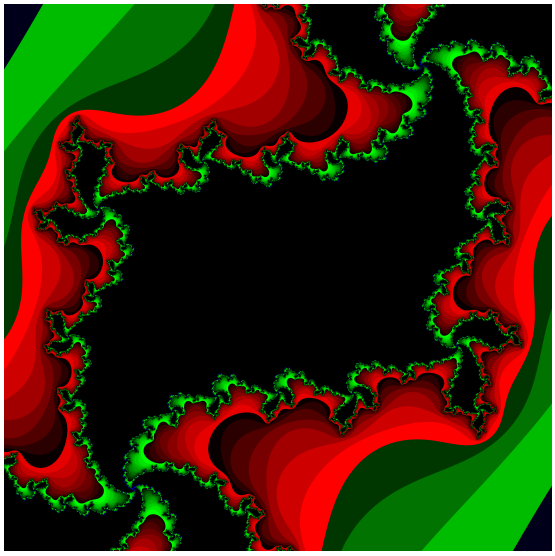

Figure 2: Changed Julia image set (diam=3.141592, centRe=-1.6, centIm=0.312)

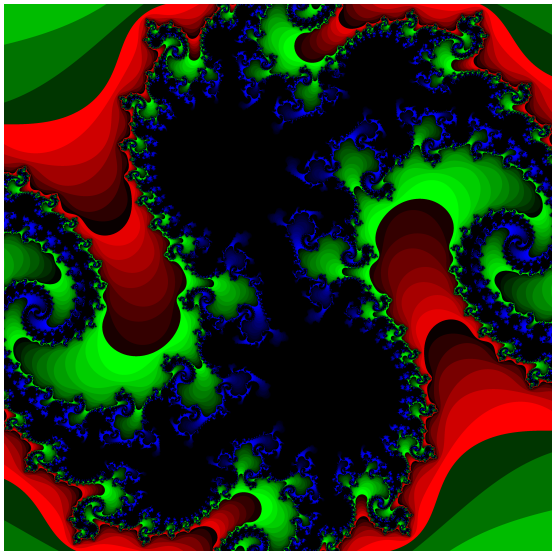

Figure 3: Original Julia image

Q3 *(15 points) Profiling CUDA code.*

Profile your CUDA Mandelbrot and CUDA Julia code from **Q1** and **Q2** using `nvprof`. Copy the output into your report. (Hint: Use LaTeX's verbatim environment.)

Mandelbrot output:

```
elapsed = 0.010000
Printing mandelbrot.png...done.
elapsed = 0.020000
Printing mandelbrot.png...done.
elapsed = 0.010000
Printing mandelbrot.png...done.
elapsed = 0.020000
Printing mandelbrot.png...done.
elapsed = 0.010000
Printing mandelbrot.png...done.
Exited script normally
```

Julia output:

```
elapsed = 0.020000
Printing julia.png...done.
```

nvprof output:

```
[kjiang@nr159 HW5]$ nvprof ./julia 4096 4096 32
==50577== NVPROF is profiling process 50577, command: ./julia 4096 4096 32
elapsed = 0.020000
Printing julia.png...done.
==50577== Profiling application: ./julia 4096 4096 32
==50577== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  21.454ms         1  21.454ms  21.454ms  21.454ms  [CUDA memcpy DtoH]

==50577== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 92.26%  271.79ms         1  271.79ms  271.79ms  271.79ms  cudaMalloc
  7.51%  22.137ms         1  22.137ms  22.137ms  22.137ms  cudaMemcpy
  0.11%  316.94us         1  316.94us  316.94us  316.94us  cuDeviceTotalMem
  0.10%  299.87us        91  3.2950us     126ns  109.52us  cuDeviceGetAttribute
  0.01%  24.940us         1  24.940us  24.940us  24.940us  cuDeviceGetName
  0.00%  3.8410us         6     640ns     188ns  2.0320us  cudaSetupArgument
  0.00%  3.0740us         1  3.0740us  3.0740us  3.0740us  cudaLaunch
  0.00%  2.6820us         3     894ns     200ns  2.0730us  cuDeviceGetCount
  0.00%  1.3320us         1  1.3320us  1.3320us  1.3320us  cudaConfigureCall
  0.00%     898ns         3     299ns     190ns     475ns  cuDeviceGet
```

Q4 *(15 points) Memchecking CUDA code.*

Run CUDA memcheck on your CUDA Mandelbrot and CUDA Julia code from **Q1** and **Q2** code. (For full credit, there should be no memory errors.) Copy the output into your report. (Hint: Use LaTeX's verbatim

environment.)

After performing a cuda-memcheck on both files, I only come up with 1 error for a call to cudaLaunch.
Mandelbrot memory check:

```
[kjiang@nr159 HW5]$ cuda-memcheck ./mandelbrot 4096 4096 32
========= CUDA-MEMCHECK
========= Program hit cudaErrorInvalidConfiguration (error 9) due to "invalid configuration argumen
=========     Saved host backtrace up to driver entry point at error
=========     Host Frame:/lib64/libcuda.so.1 [0x2ef343]
=========     Host Frame:./mandelbrot [0x376be]
=========     Host Frame:./mandelbrot [0x36b7]
=========     Host Frame:./mandelbrot [0x3306]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xf5) [0x21b15]
=========     Host Frame:./mandelbrot [0x343d]
=========
elapsed = 0.020000
Printing mandelbrot.png...done.
========= ERROR SUMMARY: 1 error
```

Julia memory check:

```
[kjiang@nr159 HW5]$ cuda-memcheck ./julia 4096 4096 32
========= CUDA-MEMCHECK
========= Program hit cudaErrorInvalidConfiguration (error 9) due to "invalid configuration argumen
=========     Saved host backtrace up to driver entry point at error
=========     Host Frame:/lib64/libcuda.so.1 [0x2ef343]
=========     Host Frame:./julia [0x3771e]
=========     Host Frame:./julia [0x3715]
=========     Host Frame:./julia [0x331f]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xf5) [0x21b15]
=========     Host Frame:./julia [0x344d]
=========
elapsed = 0.020000
Printing julia.png...done.
========= ERROR SUMMARY: 1 error
```