Important information about plagiarism:

While you are encouraged to discuss lecture topics with your peers, assignments must be your own work and plagiarism of any kind will not be tolerated. Submitted assignments will be automatically cross-checked with all other submissions in the class. Anyone who plagiarizes (i.e., copies code or allows their code to be copied by others) will receive a zero (0) on the assignment and be referred to the department chair for more severe disciplinary action.

Introduction

In this assignment, you are going to write a CUDA program to generate the Mandelbrot fractal.

A Mandelbrot set is a set of points in a complex plane that are quasi-stable (will increase and decrease, but not exceed some limit) when computed by iterating a function, usually the function used is $z_{k+1} = {z_k}^2 + c$ where z_{k+1} is the $k+1^{\text{th}}$ iteration of the complex number $z = a + bi$ (where $i = \sqrt{-1}$), z_k is the k^{th} iteration of z, and c is a complex number giving the position of the point in the complex plane. The initial value for z is zero. The iterations are continued until the magnitude of z is greater than 2 (which indicates that z will eventually become infinity) or the number of iterations reaches some arbitrary limit. The magnitude of z is the length of the vector and is given by $z_{length} = \sqrt{a^2 + b^2}$.
Computing the complex function $z_{k+1} = {z_k}^2 + c$ is simplified by recognizing that $z^2 = a^2 + 2abi + b^2i^2 = a^2 - b^2 + 2abi$, or a real part that is $a^2 - b^2$ and an imaginary part that is $2abi$. Hence, if z_{real} is the real part of z, and z_{imag} is the imaginary part of z, the next iteration's values can be produced by computing: $z_{real} = {z_{real}}^2 - {z_{imag}}^2 + c_{real}$, $z_{imag} = 2z_{real}z_{imag} + c_{imag}$ where c_{real} and c_{imag} are the real and imaginary parts of c, respectively.

Your program should work like the following:

```
./<program> <outfile> size
```

Here `size` is a value to indicate that the output image should be of `size` × `size` pixels.

To make it easier, you can assume that the output file is to be in the PGM P5 format.

The PGM P5 format

The PGM format is a lowest common denominator grayscale image file format. The format is defined as follows:

- A "magic number" for identifying the file type. A pgm file's magic number is the two characters "P5".
- Whitespace (blanks, TABs, CRs, LFs).
- The width of the image (in number of pixels), formatted as ASCII characters in decimal.
- Whitespace.
- The height of the image (in number of pixels), in ASCII decimal.
- Whitespace.

- The maximum grayscale value, again in ASCII decimal (Note that this value should not be more than 255 for our purpose, and we use 31 for our images).
- Whitespace (only one character allowed here).
- Width × height pixels, each pixel is represented by one or two bytes (for our purpose, we use only one byte for each pixel). The values of these bytes must be between 0 and the specified maximum value (inclusive). The pixels are listed starting at the top-left corner of the pixmap, proceeding in normal English reading order.

There is a set of publicly available utilities named "`netpbm`" to convert the PPM image format to and from other image formats (such as `png`, `tiff`, `jpg`). You can easily find these utilities on the Internet and install them on your own machine. There are also utilities that allow you to view PPM images directly. Examples include "OpenSeeIt" (openseeit.sourceforge.net) and "IrfanView" (www.irfanview.com – it also has a portable version – that means you do not need admin privileges to install, the portable version is available at protableapps.com). Please note that both of these tools are Windows applications, meaning that you must transfer the image from the cluster to Windows to see it – this can be done by using the SFTP function on MobaXTerm[1].

If you are connecting to the cluster with the "-X" option[2] through MobaXTerm, you can also use the xview program to view PPM images.

```
xview image.pgm
```

Note that the xview application is only available in the front node, not in any of the compute nodes.

Additional Information:

Your program must be a CUDA program. When executed, it should launch a two-dimensional grid of blocks, with 16 by 16 threads in each block. However, you cannot assume this knowledge in your kernel. That is, you must calculate the total number of threads by using `gridDim` and `blockDim`.

Several files are provided to reduce the effort on your side.

1. `image.h` defines a data structure (image) to hold an image; it also declares prototypes of the functions that you can use, and these functions are defined in `image.cu`:

```
image *create_image(int width, int height, int maxValue);
image *read_image(const char *filename);
int write_image(const char *filename, image *photo);
void clear_image(image *photo);
```

[1] A caveat is that when you are running MobaXTerm from RDS, the "Desktop", "Documents", "Downloads", … folders refer to the corresponding folders on the virtual computer running on the server located in ITS. You usually have no access to these folders. In order for you to see the downloaded file(s), you must choose to save them on a drive that is accessible from your Windows environment, usually appears in the file chooser as "C on …" etc.

[2] Note that it has to be uppercase "X". This option allows X-Server graphics to pass through the SSH connection and be displayed on your screen.

2. `image.cu` provides function definitions for functions declared in `image.h`.

   (Note: The `image.h` and `image.cu` files are NOT the same as the ones provided in Assignment 2)

3. `smandelbrot.cu` is a reference implementation of the sequential algorithm. You can compile this program using the following command[3]:

```
nvcc -o smandelbrot smandelbrot.cu image.cu
```

4. You should name your program `pmandelbrot.cu`, and you should be able to compile it using the following command:

```
nvcc -o pmandelbrot pmandelbrot.cu image.cu
```

5. When you launch the kernel, you should use a two-dimensional grid and two-dimensional blocks. Block size should be 16 by 16 threads per block. The number of blocks should be minimized (that is, you should not use more blocks than necessary). You can assume that each thread will only be responsible for calculating only one pixel value. That is, you do not have to worry about the cases that we have more pixels than number threads allowed on the device. However, your code might have to deal with the situation that the output image size may not cover all the threads that are launched. That is, there may be threads that must be left idle for the program to function correctly.

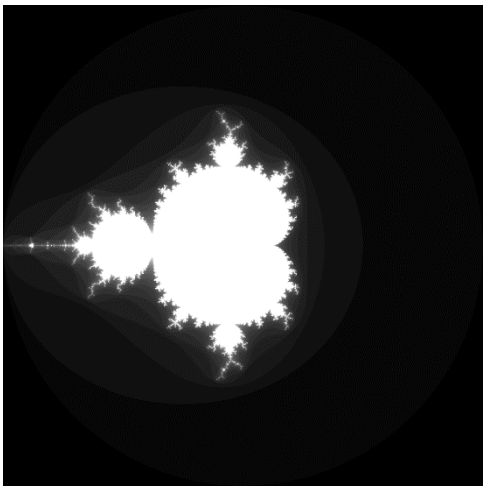
6. You can run the sequential and parallel program as follow:

```
./smandelbrot sresult.pgm 1024
```

```
./pmandelbrot presult.pgm 1024
```

   The resulting Mandelbrot image should look like the following:

[3] You should also be able to use gcc to compile this program. However, since nvcc on Linux is backward compatible with gcc, we choose to always use nvcc for consistent.



Use of AI generated contents

Light use of AI generated content is allowed. By light use, it usually means less than 10% of the stuff you submitted. Examples of such content include (but not limited to) co-pilot for code contents and Chat-GPT for non-code contents.

However, such uses must be reported. The report will be used to determine if your use of AI-generated content is light or not. If you are found using unreported AI-generated contents in your submission, the entire assignment will receive a grade of zero, and the incident will be reported to the department chair for further investigation.

In coding assignments, you should add the report at the end of your code in the form of code comments. If applicable, this will not count as fulfilling documentation requirement.

For each item you are reporting, you must include at least the following (repeat for every and all items):

- Date you use the service.
- Name of the service.
- The prompt you used.
- The response generated.
- The part of the response you have used in your submission.

Please note that you must report the use of AI-generated contents even if you have modified the contents before using it in your submission.

Example:

```
# Date: July 1, 2024
# Service: Co-pilot
# Prompt: type in "def convert_int_to_string"
# Response:
#   def convert_int_to_string(number):
```

```
#       return str(number)
# Used: the entire response
```

Submission Checklist

- ☐ Can you compile your file on the cluster without any compiler errors and/or warnings?
- ☐ Can you execute your program without any runtime errors?
- ☐ Is your program written in CUDA C?
- ☐ Do you compute the grid dimension in your kernel instead of assuming a fixed grid dimension?
- ☐ Have you included comments in your program to explain how your program works?

Submission Requirement:

Except for `pmandelbrot.cu`, you should not modify any other files provided. You only need to submit the following files:

- Your program file `pmandelbrot.cu`.

Please submit the file to Canvas – due date: Sunday October 19.