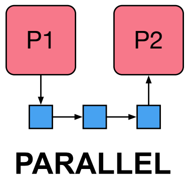
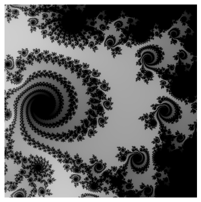


CISC 372: Parallel Computing

University of Delaware, Spring 2022

HW 7



This assignment is due at noon on ***Monday, May 16****. Topics: CUDA.*

1. Instructions

You already have a directory named `hw` in your personal repository. Add a subdirectory `hw07` to that directory.

2. Background: The Mandelbrot Set

The Mandelbrot set is the set of all complex numbers c such that the sequence

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

is bounded. See `https://en.wikipedia.org/wiki/Mandelbrot_set` for further background on this set. One of the fascinating features of this set is that it exhibits repeating patterns at every scale, i.e., it is fractal.

See program `372-2022S/code/src/seq/mandelbrot/mandelbrot.c`. This program calculates, up to a given a maximum, the number of iterations that are needed for $|z_k|^2 \geq 5$. Appropriately scaled x and y coordinates of each pixel determine the real and imaginary components of c. The number of iterations (k) determines the color of each pixel.

To make an interesting animation, after each frame, the program "zooms in" a bit towards the mid-point, by decreasing `delta` by 1%.

The program takes three command line arguments: the width, measured in pixels, which is also the height (the animation is a square); the number of time steps; and the name of the output file (an ANIM file). It uses the (now familiar) ANIM library to construct the animation file. The generated ANIM file can be converted to GIF or MPEG4 and viewed.

3. Problem: CUDA Version of Mandelbrot

In this exercise, you will write a CUDA version of the Mandelbrot program.

Your goal is to write the fastest program you can while still being correct, i.e., functionally equivalent to the original sequential code.

Work in subdirectory `hw07/mandelbrot`. Call your CUDA program `mandelbrot.cu`.

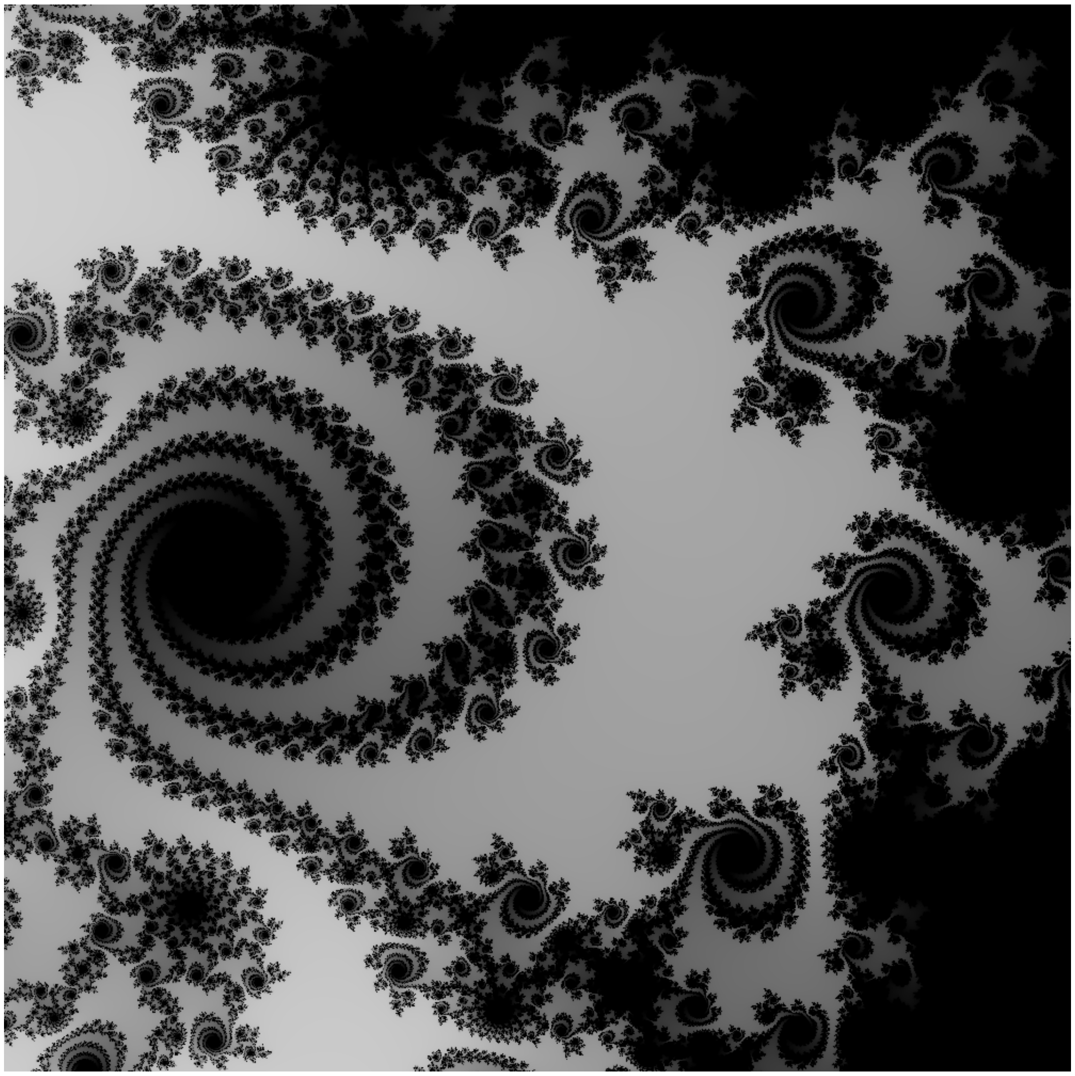


FIGURE 1. Image of Mandelbrot set generated by `mandelbrot.exec 1000 1` and converted to GIF via `anim2gif -color gray mandelbrot.anim`.

There are a few technical points to keep in mind. First, CUDA-C is slightly different from standard C. (It actually has more in common with C++.) For example, array parameters in function prototypes or definitions cannot have lengths. For this reason, a CUDA version of the ANIM header file is provided; it is called `cudaanim.h`. This is the file you should include instead of `anim.h`.

Second, CUDA-C does not support literal arrays and structs. You will therefore have to declare variables and initialize them, and use these variables as arguments to the ANIM functions, rather than using literal arrays as we have done in the past. A nice example illustrating correct use of ANIM in a CUDA-C program is `diffusion2d.cu`, which is committed to our class repository.

Third, while floating-point operations on doubles should be identical on the CPU and GPU, there is one way in which results may differ: the GPU has a "fused multiply-add" instruction which allows it to evaluate an expression such as `a*b+c` in a single operation. This is not only faster than using two separate arithmetic instructions, but also possibly more accurate. Because of this, you may get slightly different results. When simulating a chaotic phenomenon such as the fractal computation, a slight difference can explode into a very large difference after a few iterations. So don't be surprised if you see that!

How then do you check your code? Simple: you can tell the CUDA compiler to turn off fused multiply add with the command line flag `--fmad false`. Using that flag, you should get the exact same output from the CUDA and the original sequential CPU programs. A `Makefile` is provided to help you check this; it can be used to compile code on Grendel or Bridges-2 and can be used to run tests on Grendel/Beowulf. (For Bridges-2, you must use batch files.)

Once you have your program working and have verified its correctness, it is time for a simple performance experiment. Use one GPU-shared node on Bridges-2, and run `mandelbrot.cu` with a width of 1200 pixels and 400 time steps. Call your batch script `mb_cuda.sh` and the corresponding SLURM output `mb_cuda.out`. Commit both of these in `hw07/mandelbrot/bridges`.

In a short `README` file in `hw07/mandelbrot` place the following information:

```
Sequential time (s): XXX.X
CUDA time (s): XXX.X
Speedup: XX.X
```