Bo Pang (pangb@berkeley.edu)
01/22/2021

CS267 – Project Pre-proposal:

Fractal Art

About me

Hi, my name is Bo Pang. I am a 1st year MEng EECS student at UC Berkeley. My primary concentration of studying is embedded systems. Prior to this, I had some industry experience with enterprise distributed storage systems, where I worked on software that abstracts and manages storage hardware.

I enjoy having software that utilizes hardware potentials – such as 8K video playback via GPU's HEVC decoder; data encryption made by the drive's onboard ASIC or Intel's AES-NI. I believe I will learn about this exciting field in much more depth by taking this class.

Application – Fractal Art (e.g., Mandelbrot set)

Fractal arts are visually appealing but computational intense to generate. There are many kinds of fractal art; in the following discussion, I take its famous example – **Mandelbrot set**. See *Figure 1.*

Overview

Given an iterative algorithm on complex number *z* and *c*: $z_{n+1} = z_n^2 + c \ \ (with\ z_0 = 0)$, the Mandelbrot set is the set of *c* that z_n does not diverge. To generate the plot, we try run the algorism on each *c* (i.e., each pixel) located in the complex plane (i.e., our picture frame), and color each *c* depends on how the corresponding *z* changes over the iterations (e.g., black=not diverge; light/dark=fast/slow diverge).

For a formal & better description, see the wiki page [1].

We can also "zoom in" the Mandelbrot set plot by re-generating the plot on a smaller range of *c* values with higher resolution.

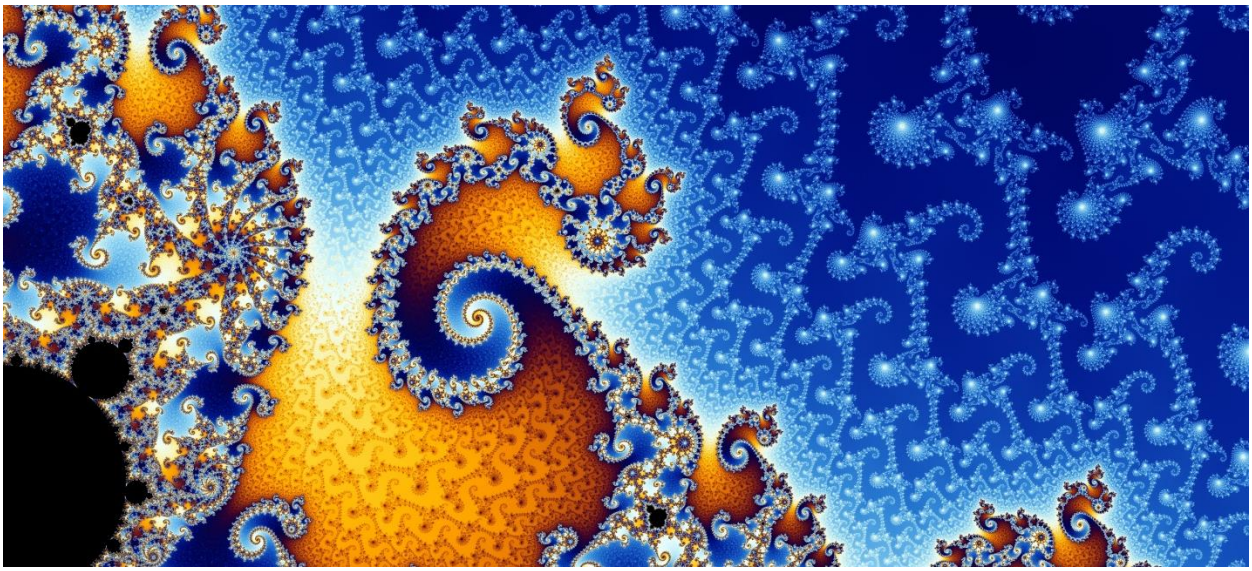

Figure 1: A small "zoomed-in" section of Mandelbrot set plot.

Bo Pang (pangb@berkeley.edu)
01/22/2021

Opportunities for Parallel Computing

There are several potential approaches to parallelize the Mandelbrot set generation.
When generating a plot, there will be a lot of *c* (depending on the plot size and resolution) we need to calculate; and each calculation is rather independent. So:

- Obviously, we can multi-thread each *c*'s calculation and gather calculation results on a shared data structure.
- We can also utilize GPU and interface like CUDA to parallelize the calculation of each *c.*

With-in the calculation of each *c*, there might be opportunities to speed up, such as:

- Use SIMD instructions to handle the complex value arithmetic by manually break apart the real & imaginary parts.
- Use FMA functions to do the "multiplication and add" calculation.

There is also some research work exploring faster generation of fractal arts. Article [2] is a relevant work that also gives a nice overview of Mandelbrot set generation.

Challenges

One of the challenges is 'branching' ability. To calculate a specific *c*, we might want to stop the iteration early (e.g., when the calculation soon converges to a number) and save time. But if we parallelize the calculation of multiple *c* using SIMD or CUDA, they must undergo the same number of iterations. That seems like a trade-off.

Another challenge relates to precision. Floating-point precision is critical for the Mandelbrot set's accuracy, especially when we "zoom-in" the plot. We can use double-precision values, or even "Quadruple-precision" values with some compiler's help. But is that supported by SIMD instructions on Cori's CPU? Is that efficient on GPUs?

Anyway, I think fractal art generation is an interesting topic and is in-scope for this class as a final project.

References

[1] https://en.wikipedia.org/wiki/Mandelbrot_set

[2] L. Jonckheere. *Efficiently generating the Mandelbrot and Julia sets.* URL: https://theses.liacs.nl/pdf/2018-2019-JonckheereLSde.pdf