Final Project Proposal: Parallelizing the Mandelbrot Set

Team Members:
Max Zhang (mz2956)
Isabel Tu (it2334)
Project Name: `par-fractal`

Project Overview

The Mandelbrot Set is a well-known mathematical set of complex numbers, defined by the iterative formula:

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

where c is a complex number and $z_0 = 0$. A point c belongs to the Mandelbrot Set if the sequence of z_n remains bounded (i.e., does not tend to infinity) as $n \to \infty$.

Visualizing the Mandelbrot Set involves plotting each point in the complex plane and coloring it based on the number of iterations needed for $|z_n|$ to exceed a threshold (usually $|z| > 2$) or by determining whether it belongs to the set. The result is a fractal image, where points inside the set are typically colored black, and points outside are colored according to their "escape time."

This project will focus on parallelizing the computation of Mandelbrot Set images, leveraging Haskell's parallel programming constructs. If time permits, we will extend the implementation to render related fractals, such as the Julia Set.

Generating Mandelbrot Set Images

To generate an image of the Mandelbrot Set:

1. **Complex Plane Sampling:** Define a grid of points in the complex plane within a chosen range (e.g., $[-2.0, 1.0]$ for the real axis and $[-1.5, 1.5]$ for the imaginary axis). The resolution of the grid determines the image quality.

2. **Iteration and Escape Calculation:** For each grid point (complex number c):

   - Iterate the formula $z_{n+1} = z_n^2 + c$ up to a maximum number of iterations.
   - Determine whether z_n remains bounded or escapes.
   - Assign a color to the point based on its escape time or boundedness.

3. **Rendering:** Output the computed data as an image.

Parallelization Strategy

We propose the following parallelization strategies using Haskell:

Using `Control.Parallel` and `Control.Parallel.Strategies`

- Divide the grid into chunks (e.g., rows or blocks).
- Use `par` and `pseq` to evaluate chunks in parallel.
- Employ `parListChunk` with a suitable evaluation strategy to distribute computation across cores.

Leveraging `REPA` (Regular, Shape-Polymorphic Parallel Arrays)

- Represent the grid as a multidimensional array.
- Use `REPA` to compute escape times in parallel.
- Take advantage of `REPA`'s efficient array manipulation and computation capabilities.

Exploring `Accelerate` for GPU Computation

- If time allows, experiment with `Accelerate` to offload computation to a GPU.
- Represent the grid as an array and implement the iteration logic in a GPU-compatible manner.

Input Data Plan

The input to the algorithm will be:

- **Grid Parameters:** Resolution (e.g., 1920×1080), real and imaginary ranges, and maximum iterations.
- **Output File Format:** The computed data will be written to an image file format (e.g., PNG or BMP) using Haskell libraries like `JuicyPixels`.

For the Julia Set, additional parameters such as the constant c for the iteration will be required.

Goals and Scope

1. **Deliverable:**
    - Render the Mandelbrot Set using parallel computation with `Control.Parallel` and `Control.Parallel.Strategies`.
2. **Stretch Goals:**
    - Implement the algorithm using `REPA` and/or `Accelerate`.
    - Extend the implementation to render the Julia Set.
    - Explore visual enhancements such as coloring schemes and high-resolution outputs.

References

1. `https://complex-analysis.com/content/mandelbrot_set.html`