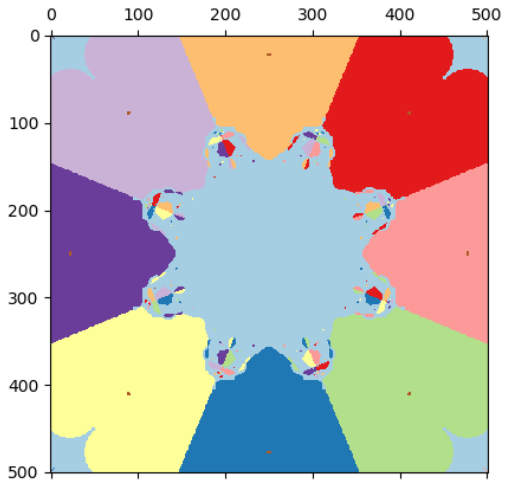
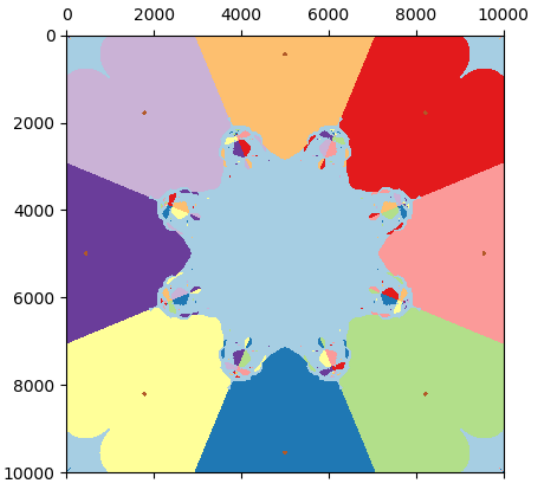
MCS 572 Project One: computing fractal plots

Message Passing to distribute root finding jobs

The goal of the project is to apply message passing to make fractal plots faster. As an example we visualize the basins of attraction of the method of Muller. The method of Muller is an iterative method to compute roots of a polynomial in one variable.

In the plots below, the method is applied to $p(x) = x^8 - 1$. The plotted square at the left represents the complex plane for both real and imaginary parts in $[-1.1, +1.1]$. The plot at the right is a magnification of a square in the plot at the right. Each point in the plot corresponds to a complex number z_0 used as the start of the iteration in the method of Muller. The color of z_0 is determined by the root computed by the iteration. In the plot below, the red regions contains all z_0 for which the method converged to the root $\sqrt{2}/2 + I\sqrt{2}/2$, where $I = \sqrt{-1}$. Points close by a root will converge to that root and we might expect to see a Voronoi diagram, as is indeed the case for points around the complex unit circle. But, for z_0 for which $|z_0|$ is small, the convergence is rather unpredictable and we obtain fractal regions.





The plot at the left was made with the Muller method in Python, the plot at the right was done with Julia.

Let $[a, b]$ be the interval for the real parts and the imaginary parts of the point z_0 at which to start the method of Muller. For dimension n, let $\Delta z = (b - a)/(n - 1)$. Then the grid of points is defined by the loops

for k from 1 to n do
  for ℓ from 1 to n do
    $z_0 = (a + (k - 1)\Delta z) + (a + (\ell - 1)\Delta z)I$

The plots are for $a = -1.1$ and $b = +1.1$. For $n = 10001$, the plots require about 100 million calls to the method of Muller. This takes several minutes for the C code (compiled with `-O3`), and a little longer with the Julia program. Because Python is so much slower, take $n = 501$.

For $n = 10001$, the file with all integers used in the visualization occupies about 200MB.

To help you start the project, three programs are posted at the course web site, `muller.c`, `muller.jl`, and `muller.py`, respectively in C, Julia, and Python. The Python script uses `matplotlib` for the rendering of the plot. The C and Julia program write comma separated integers at the end. Redirecting the output of those programs to a text file, which then serves as the input to the script `matrixplot.py`.

Assignment One: define the message passing

Write a parallel program to define the message passing for a program using MPI in a static work load assignment. The root node is responsible for assigning which regions what worker nodes compute and for writing the final matrix.

Assignment Two: examine the scalability

Investigate scalability when more worker nodes are available. Take the size of the problem into consideration. How large should the problem size be with 2, 4, 8, 16, 32, 64 nodes? Determine the granularity of the parallel code.

Determine experimentally the relation between the communication and the computation cost to determine the scalability of the program developed in the first assignment. In your cost considerations, take the memory usage into account because memory could become a bottleneck for a serial program.

Assignment Three: discuss load balancing

In some regions, the method of Muller will converge faster than in other regions. Thus some regions may require much less work than others. Describe a situation where static work load assignment does not perform very well.

Develop an algorithm for dynamic load balancing to apply for any number of processors. Decide on the granularity of the size of each job and motivate your choice. Indicate the application of the dynamic load balancing code we have discussed in class to this problem.

The deadline is Friday 27 September 2024 at noon

Submit your solution to gradescope.

You may work individually or in pairs on this project. For each pair, please submit only one solution.

If you have questions or difficulties with the assignments, feel free to come to my office for help.

Appendix. The Method of Muller

Generalizing the secant method, Muller's method is a three point iterative method of the form

$$x_{k+1} = g(x_k, x_{k-1}, x_{k-2}), \quad k = 0, 1, \ldots$$

which we will apply to approximate roots of a polynomial $f(x) = 0$.

The secant method computes x_{k+1} as the point where the line passing through $(x_k, f(x_k))$ and $(x_{k-1}, f(x_{k-1}))$ meets the horizontal axis. Muller's method constructs a parabola p through the points $(x_k, f(x_k))$, $(x_{k-1}, f(x_{k-1}))$, and $(x_{k-2}, f(x_{k-2}))$. Using the quadratic formula applied to $p(x) = 0$, x_{k+1} is set as the (complex) root of p that is closest to x_k. Naturally, if $\deg(f) = 2$, then one step with Muller's method suffices to compute one root.

The parabola passing through $(x_k, f(x_k))$, $(x_{k-1}, f(x_{k-1}))$, and $(x_{k-2}, f(x_{k-2}))$ is constructed as

$$p(x) = f(x_k) + d_1(x - x_k) + d_2(x - x_k)(x - x_{k-1})$$

with

$$d_1 = \frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}, \quad d_0 = \frac{f(x_{k-2}) - f(x_{k-1})}{x_{k-2} - x_{k-1}}, \quad d_2 = \frac{d_1 - d_0}{x_k - x_{k-2}}.$$

Writing $p(x) = ax^2 + bx + c$, we compute $d = \sqrt{b^2 - 4ac}$. Then

$$\Delta x = -2c/e, \quad \text{where} \quad e = \max(|b - d|, |b + d|),$$

is the update to the closed root to x_k, as $x_{k+1} = x_k + \Delta x$.

In Julia, we use `evalpoly` to evaluate a polynomial `p` at `x` as `y = evalpoly(x, p)`. A polynomial `p` of degree d is represented by an array of $d + 1$ numbers, starting with the constant term. So $\mathtt{p} = [\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3]$ represents the parabola $\mathtt{p}_3 x^2 + \mathtt{p}_2 x + \mathtt{p}_1$. The default coefficient type in this project is the 64-bit floating-point complex number `Complex{Float64}`.

The function `muller` is available in the file `muller.jl` posted at the course web site. Versions in C and Python are respectively in `muller.c` and `muller.py` at the course web site.