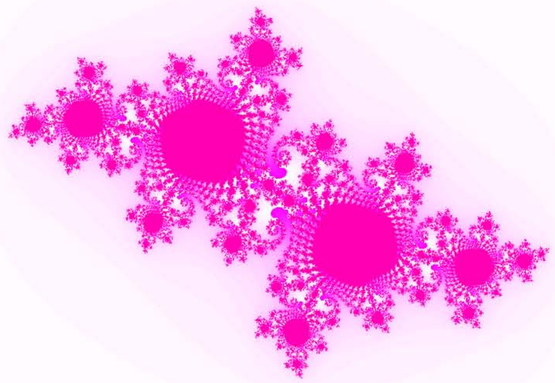
MPI Programming for

Julia Sets

Wesley Crick
Kanwar Singh

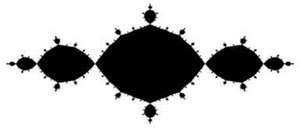
CP400N (Introduction to Parallel Programming) Term Project
Winter 2014

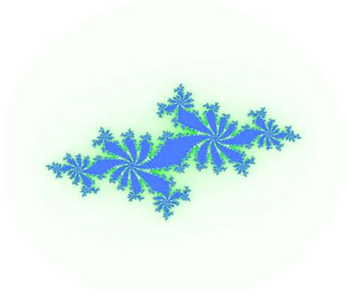
Dr. Ilias Kotsireas
Wilfird Laurier University
Physics and Computer Science Department

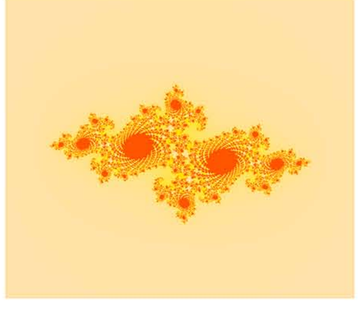


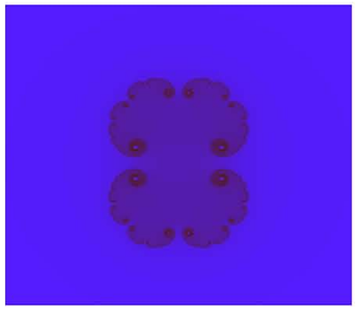
Introduction

The Julia set is named after the French mathematician Gaston Julia. The Julia set is a set of complex numbers that are then mapped to the real plane. The points in the Julia set will escape to infinity, however if wrapped around an orbital there will be two set of points, ones that escape to infinity and ones that do not. The images show the points that do not escape. The colors are decided based on how far the points are outside of the orbital.









Complex plane to real plane

The Julia set is calculated with complex numbers and then they are moved, or mapped, to the real plane. An example of this is (x + iy)(x + iy) = x^2 - y^2 + i(2xy). Then we say z = sqrt(x^2 + y^2), where x is the real part and y is the imaginary part given in the equation above. Then the Julia set is Qc(z) = z^2 + c where c is a complex number.

To map a point from the complex plane to the real plane uses this equation:

x = x_min + k * ((x_max - x_min) / x_res)
Where:
- x is the real point
- x_min is the minimum complex x
- x_max is the maximum complex x
- k is the counter for the x iteration loop
- counts from 0 to x_res
- x_res is the resolution of the screen

y = y_min - j * ((y_max - y_min) / y_res)
Where:
- y is the real point
- y_min is the minimum complex y
- y_max is the maximum complex y
- j is the counter for the y iteration loop
- counts from 0 to y_res
- y_res is the resolution of the screen

Psudocode for Julia Sets

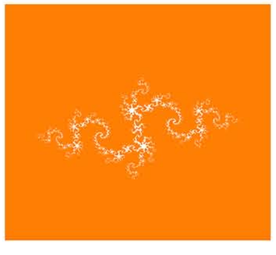
```
Set x_resoltion and y_resolution
Set max iterations
Set c1 and c2 based off of complex equation (ex. -0.1+0.8i where
c1=-0.1 and c2=0.8)
Calculate constants (dx,dy) to map from complex to real plane
(x_max- x_min/ x_resolution)

Initialize MPI
Get processor rank
Get total number of processors

Split up y_resolution among processors and loop through those
y-values
        For each y-value loop through each x point in x_resolution
                Set n=0 where n is the count to max iterations
                Calculates the imaginary point to real point

                Keep looping through until n reaches max iterations or
                escapes the orbit

                Set colour depending on if all points lie in orbit or not
                Print out points and colour for drawing program

Shut down MPI
```

Conclusion

From these results, it can be sees that there is some speedup from running Julia sets in parallel. However, the speedup that comes from running the program in parallel is very small. The time it took to compute a Julia set in serial was averaged around 1 microsecond, where at 128 processors it took, on average, .2 microseconds, which is not a huge amount. This showed that processors today are quick and efficient. In Chaos, Fractals, and Dynamics it was stated that with 100 iterations it could take hours to finish a Julia set, but when ran with 250 iterations, in serial, and it was completed in under a microsecond. Parallel computing is able to increase efforts when solving many types of problems, but it was shown that it does not significantly increase efficiency in this case. If it was run at a higher resolution, or with more iterations, then a more significant increase in speed would have been seen.

Results

Processors	Average Time(s)
1	0.000000953674
2	0.000000596046
4	0.000000526837
8	0.000000506639
16	0.000000447035
32	0.000000266067
64	0.000000214496
128	0.000000202633



References

Devaney, Robert L. "5-7." Chaos, Fractals, and Dynamics: Computer Experiments in Mathematics. Menlo Park, CA: Addison-Wesley Pub., 1990. N. pag. Print.