# Visualization of Julia Sets corresponding to different recursive functions using CUDA

**Dhruv Kohli**
**B.Tech Mathematics & Computing**
**Pre-final year**
**120123054**

# INTRODUCTION

Consider a recursive function z(n+1)=f(z(n)) where z(n) is a complex number i.e. belongs to **C** and hence f is a function from **C** to **C**. Also, denote the Julia Set corresponding to f as J(f).

Let us seed the given recursive function with z(0) = x + iy where x,y E **R** and find out the value of z(n) for a sufficiently large value of n.

If abs(z(n)) > MAX_ABS_Z (for an appropriate value of MAX_ABS_Z) then the corresponding seed z(0) is NOT in J(f) otherwise z0 is in J(f).

Through this assignment we'll consider a matrix with each cell mapped to a certain complex number z(i,j) and a recursive function f:**C**->**C**. Then we'll simply calculate whether z(i,j) belongs to J(f). If z(i,j) is in J(f) then we'll color the cell in the matrix corresponding to z(i,j) BLACK else we'll color it based on the value of 'n' at which abs(z(i,j)) becomes greater than MAX_ABS_Z.

Since, checking of whether z(i,j) is in J(f) is independent of checking of whether z(i',j') is in J(f), we can check for each complex point in parallel.

# CODE

Following is the kernel code for the formation of fractal(Julia Set) in CUDA.

```
__global__ void fractalForm(unsigned char *mat, int maxZAbs, int maxN, int minN, int decayN, double iReal,
                            double iImg, int categ, double rMin, double rMax, double iMin, double iMax,
                            int H, int W, int numCols) {
    int idx = threadIdx.x + blockIdx.x*blockDim.x;
    if(idx >= H*W)
        return;
    int i_ = idx/W;
    int j_ = idx%W;
    double re = rMin + (i_*(rMax-rMin))/(1.0*H);
    double im = iMin + (j_*(iMax-iMin))/(1.0*W);
    complex<double> z(re,im);
    complex<double> c(iReal, iImg);
    size_t n;
    for(n = maxN; n >= minN && abs(z) < maxZAbs; n-=decayN) {
        z = getFuncVal(z, c, categ);
    }
    rgb col = getColor(n/decayN, numCols);
    mat[3*(j_ + i_*W)]=(unsigned char)(int)col.g;
    mat[3*(j_ + i_*W)+1]=(unsigned char)(int)col.r;
    mat[3*(j_ + i_*W)+2]=(unsigned char)(int)col.b;
}
```

# Time Complexity Analysis

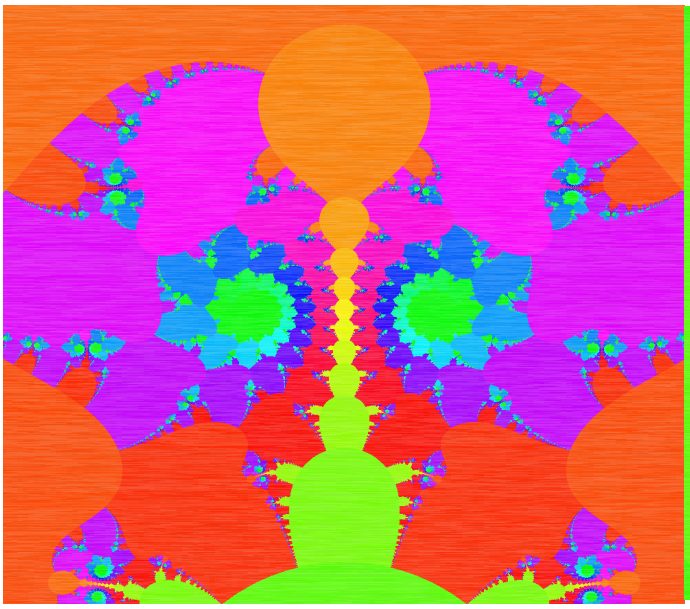
Let M be the max number of iterations used to check whether z(i,j) is in J(f).
For a matrix of size nXn, the sequential algorithm takes O(n*n*M) time.
Given infinite number of processors the parallel algorithm takes O(M) time.
Given p number of processors the parallel algorithm takes O((n*n*M)/p) time.

Therefore,

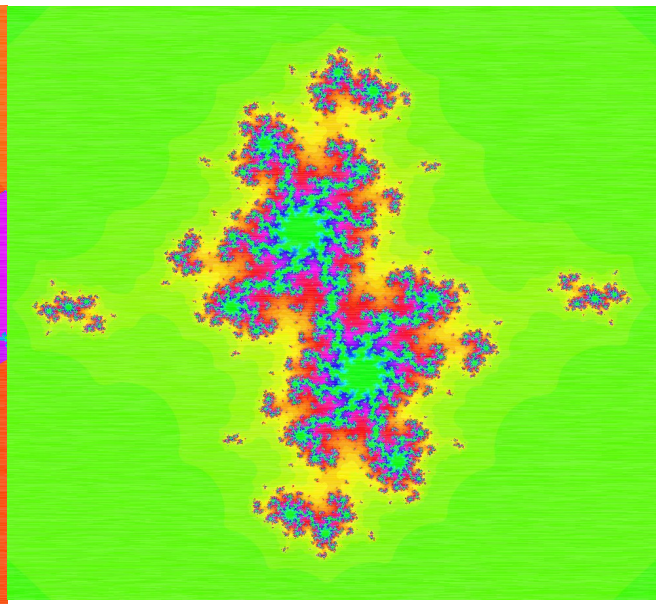
T(1) = O(n*n*M)
T(INFINITY) = O(M)
T(p) = O((n*n*M)/p)

# Results

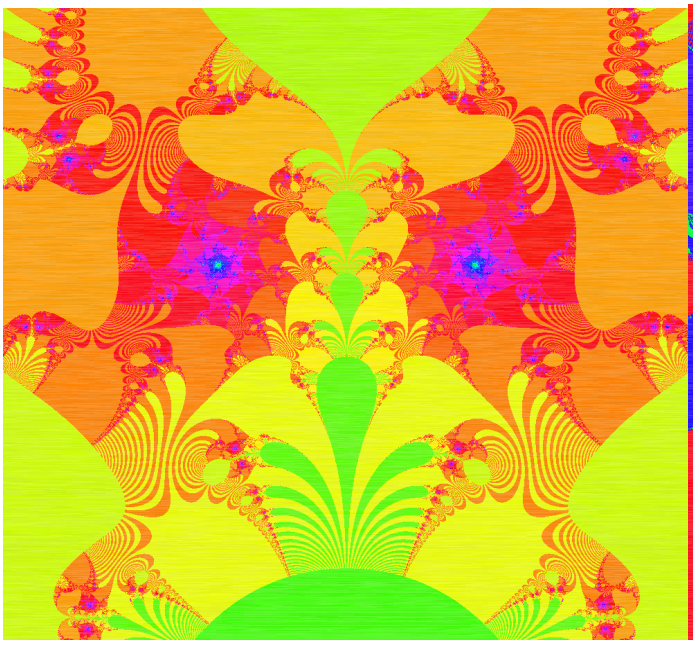
Following are some of the generated fractals or Julia Set images with the corresponding recursive function used to generate them:



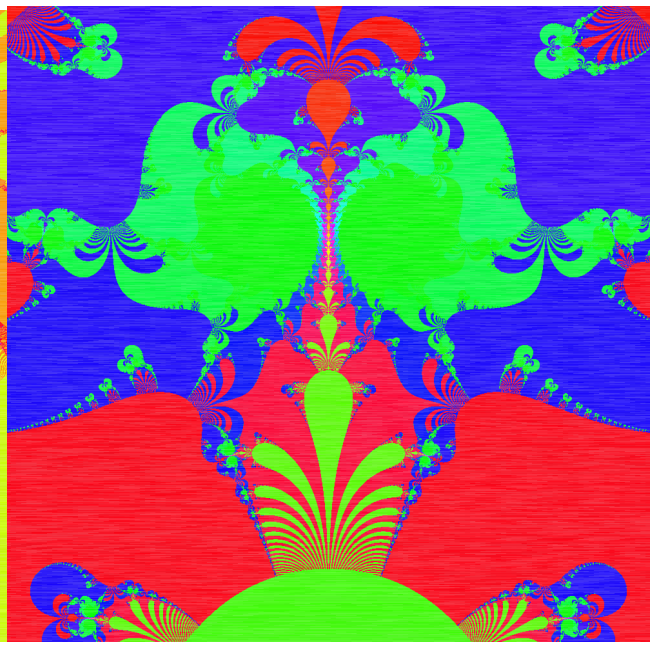
f(z)=exp(z)+c



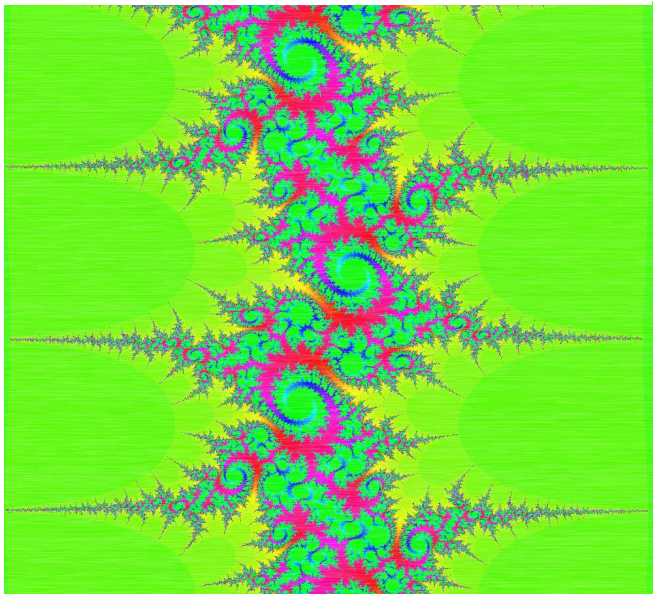
f(z)=sin(z)*z+c



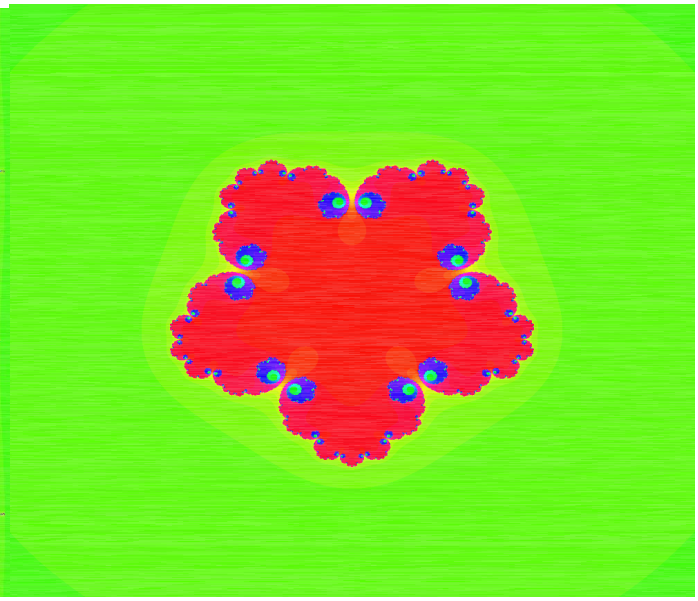
f(z)=exp(exp(z))+c1



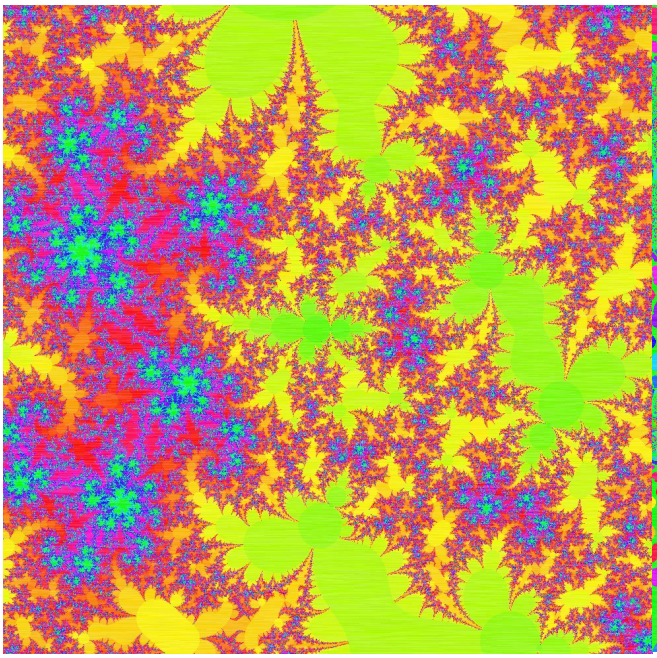
f(z)=exp(exp(z))+c2



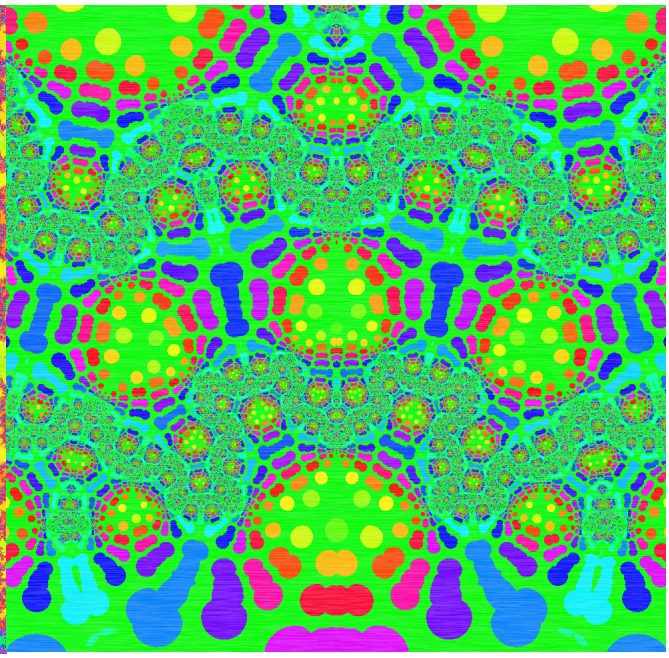
f(z)=cos(z)+c



f(z)=z^5+c



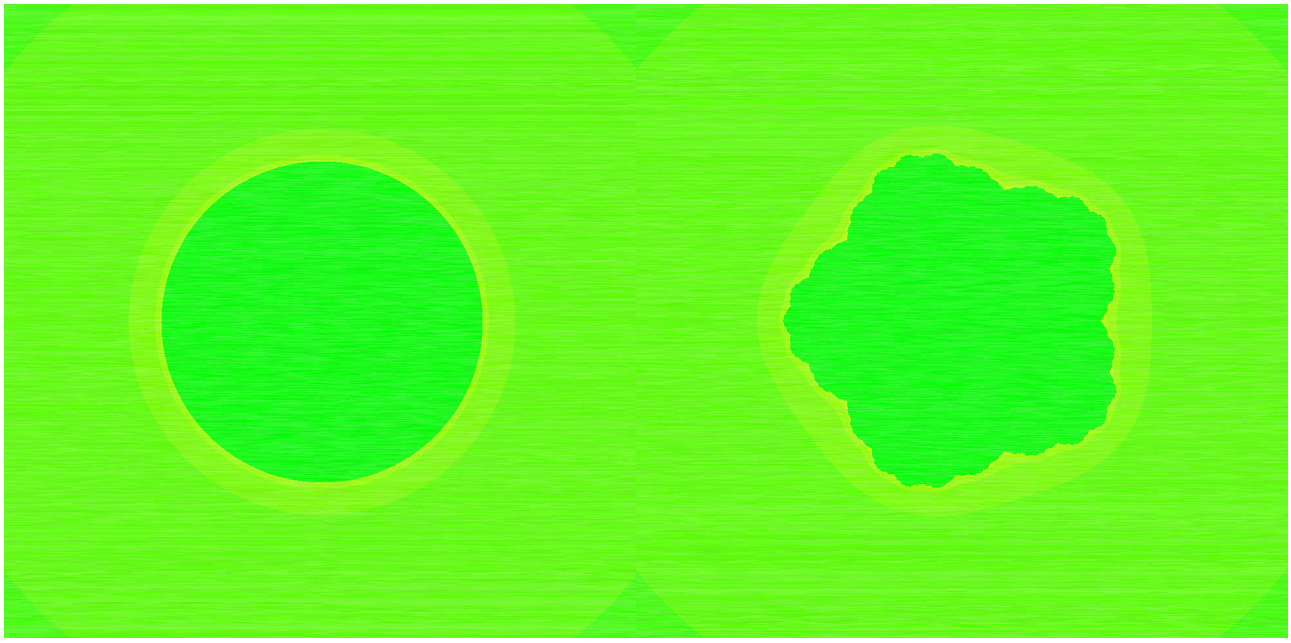
f(z)=cos(z)/(z)+c



f(z)=log(z)/(z)+c

## Time comparison (sequential vs parallel)

To reach from (a) to (b), the sequential code took 83 seconds while the parallel code took 0.39 seconds.
- Image size was 1000X1000, recursive function used was f(z)=z^5+c, #iterations to decide membership in J(f) equals 25 and MAX_ABS_Z equals 10.
- We tested on Tesla K40 machine with a launch of 1000000 threads.(1 thread per pixel of image)



(a) (b)

## Conclusion

**Fractals made using the concept of Julia Set, form the base of computer graphics. The assignment shows the tremendous speed up that we obtain in the formation of Julia Set images using a GPU as compared to a CPU. Infact, that's why the GPU's are traditionally named Graphical Processing Unit as they provide tremendous boost to the graphics computations.**

**Speed up obtained in our case = n*n where n is the size of the matrix.**

**To know more about the world of fractals:**
**1. http://en.wikipedia.org/wiki/Fractal**
**2. https://www.youtube.com/watch?v=s65DSz78jW4**

**A competition called Revision takes place in Germany, every year, in which the computer graphics programmer showcase or as their site says showoff their programming skills by making incredible programmatically generative videos. Fractals form the base of their skills (like a Hello World program in C). Have a look at this link:**

http://gizmodo.com/this-incredible-animation-was-made-by-code-that-could-f-1565294456