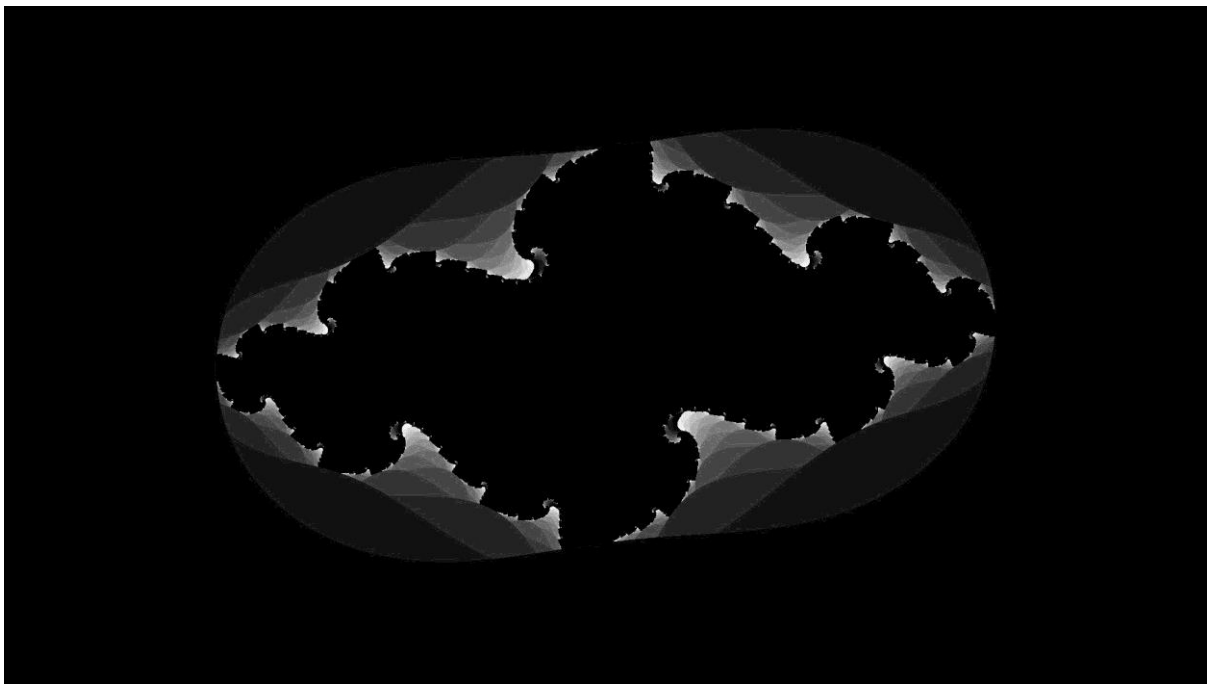
Lab 10 Additional practice with OpenMP

The serial program Listing 1 will result in this fractal picture:



Convert the program in Listing 1 into a parallel Open MP program. Only one process should do I/O, while the main calculation should be split up among all the processes.

Your Open MP version should result in the same graph regardless of how many processes we use. It should also spread the work as evenly as possible between the processes and minimize the amount of time processes would need to wait for each other.

Listing 1

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

float params[]={0, 1920,1080,-0.70174,0.156,1.2,500};

#define TEMPFILE "temp.pgm"
#define MAXPIXEL 16
int ** picture;
void allocatePicture(int width, int height);
#define putpixel(x,y,value) picture[y][x]=value
void displayPicture(int width, int height);

typedef struct{
	double x,y;
}complex;

complex add(complex a,complex b){
	complex c;
	c.x = a.x + b.x;
	c.y = a.y + b.y;
	return c;
}
```

```
complex sqr(complex a){
	complex c;
	c.x = a.x*a.x - a.y*a.y;
	c.y = 2*a.x*a.y;
	return c;
}

double mod(complex a){
	return sqrt(a.x*a.x + a.y*a.y);
}

complex mapPoint(int width,int height,double radius,int x,int y){
	complex c;
	int l = (width<height)?width:height;

	c.x = 2*radius*(x - width/2.0)/l;
	c.y = 2*radius*(y - height/2.0)/l;

	return c;
}

void juliaSet(int width,int height,complex c,double radius,int n){
	int x,y,i;
	complex z0,z1;

	for(x=0;x<width;x++)
		for(y=0;y<height;y++){
			z0 = mapPoint(width,height,radius,x,y);
			for(i=1;i<=n;i++){
				z1 = add(sqr(z0),c);
				if(mod(z1)>radius){
					putpixel(x,y,i%MAXPIXEL);
					break;
				}
				z0 = z1;
			}
			if(i>n)
				putpixel(x,y,0);
		}
}

int main(int argC, char* argV[])
{
	int width, height, iterations;
	complex c;
	float radius;

	width = params[1];
	height = params[2];
	c.x = params[3];
	c.y = params[4];
	radius = params[5];
	iterations = params[6];

	allocatePicture(width,height);
	juliaSet(width,height,c,radius,iterations);

	displayPicture(width,height);
}
```

```
void allocatePicture(int width, int height)
{
      picture = (int **)malloc(sizeof(int*)*height);
      for (int i = 0; i<height; i++)
            picture[i]=(int*)malloc(sizeof(int)*width);
}

void displayPicture(int width, int height)
{     int x,y;
      FILE *outfile = fopen(TEMPFILE,"w");
      fprintf(outfile,"P2\n%d %d %d\n",width, height, MAXPIXEL);
      for (y=0; y<height; y++)
            for (x=0; x<width; x++)
                  fprintf(outfile,"%d\n", picture[y][x]);
      fclose(outfile);
      system("display "TEMPFILE);
}
```

Note that this program uses the “display” command to display the picture. You can get this command either by installing ImageMagick or GraphicsMagick. In Ubuntu, that would be:

```
sudo apt install graphicsmagick
```

or

```
sudo apt install imagemagick
```

Alternatively, you could just open the `temp.pgm` file in a Windows picture displaying program if you prefer.