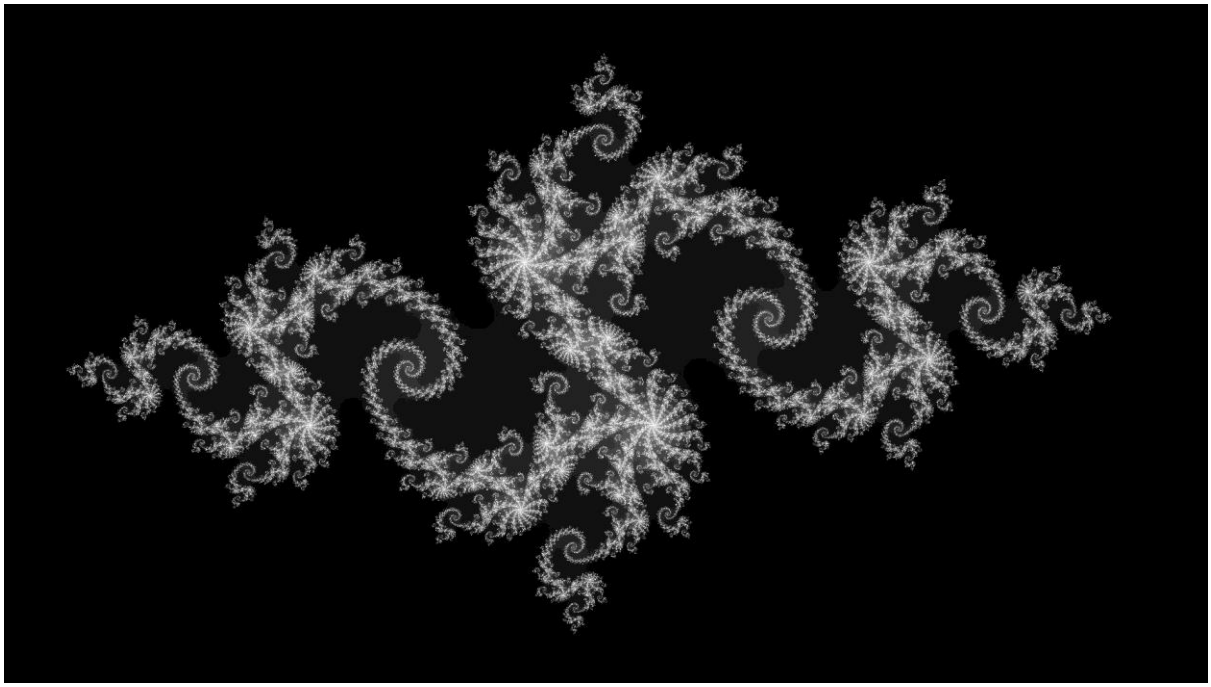
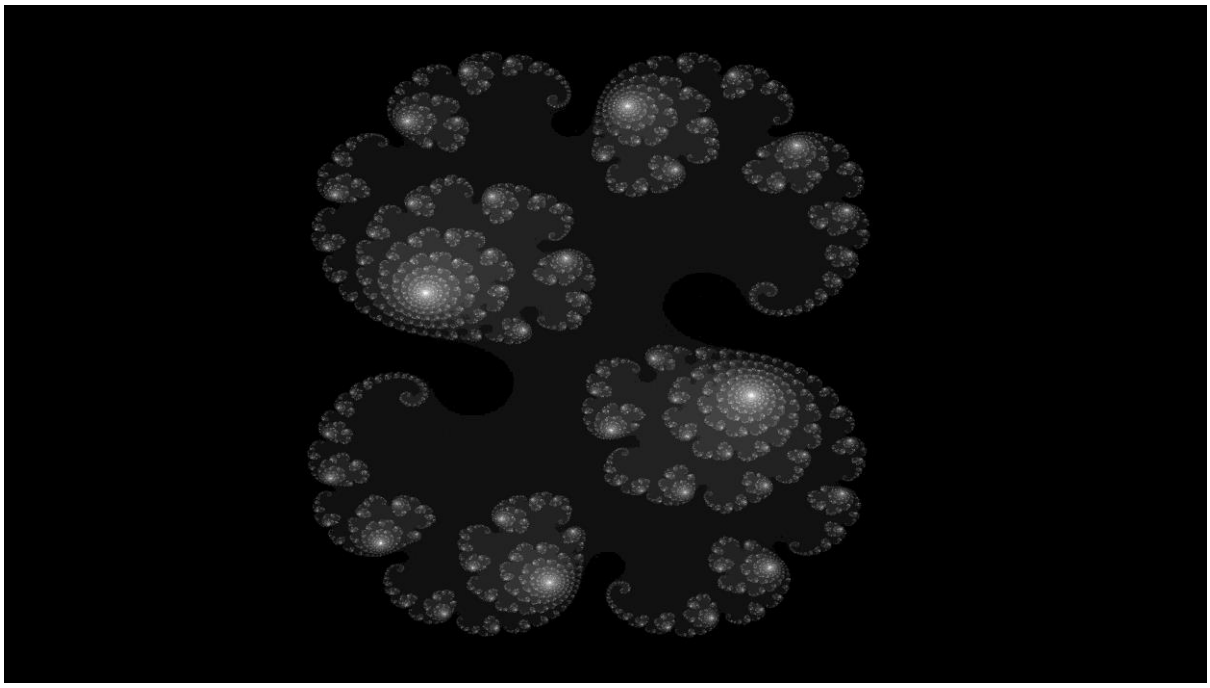
Tutorial on 2D Quadrature

Sometimes what you need to compute is not in a single-dimensional array. This tutorial will deal with this problem.

The Julia fractal is a family of beautiful fractals that can be computed mathematically. Here is an example of a Julia fractal where the Julia constants are -0.79 and 0.15, x scale & offset is 3.5 and 1.7 and y scale and offset are 2.0 and 1.0:



Here is an example of a Julia fractal where the Julia constants are 0.285 and 0.01, x scale & offset are 3.5 and 1.7, y scale and offset are 2.5 and 1.25:



For more information, and other Julia constants that have pretty results, Google “Julia fractal”.

Here is serial code to compute this fractal:

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

#define BUFSIZE 256

int xsize, ysize, maxIteration;
double cx, cy, xscale, xoffset, yscale, yoffset;

int julia(int x, int y)
{ double zx = (double)x/xsize *xscale -xoffset;
  double zy = (double)y/ysize *yscale -yoffset;

  int iteration = 0;

  while((iteration<maxIteration) && (zx * zx + zy * zy < 4))
  {
        double xtemp = zx * zx - zy * zy;
        zy = 2 * zx * zy + cy;
        zx = xtemp + cx;

        iteration++;
  }

  return iteration;
}

int main(int argc, char*argv[])
{ int x, y;
  int **picture;
  char buff[BUFSIZE];
  FILE *outfile;

  printf("Program to generate a Julia fractal\n");
  printf("What file should I store the picture in? [E.g. julia.pgm]
");fflush(stdout);
  fgets(buff,BUFSIZE,stdin); buff[strlen(buff)-1]=0; // Remove the ending
line feed
  printf("How wide and tall should the picture be? [E.g. 1920 1080]
");fflush(stdout);
  scanf("%d %d", &xsize, &ysize);
  printf("What is the x scale and x offset? [E.g. 3.5 1.7]
");fflush(stdout);
  scanf("%lf %lf", &xscale, &xoffset);
  printf("What is the y scale and y offset? [E.g. 2.0 1.0]
");fflush(stdout);
  scanf("%lf %lf", &yscale, &yoffset);
  printf("What are the x and y components of the Julia constant? [E.g -
0.79 0.15] ");fflush(stdout);
  scanf("%lf %lf", &cx, &cy);
  printf("How many times to iterate? [E.g. 512] "); fflush(stdout);
  scanf("%d", &maxIteration);

  picture =(int**)malloc(sizeof(int*)*ysize);

  for (y=0; y<ysize; y++)
  { picture[y] = (int*)malloc(sizeof(int)*xsize);
    for (x=0; x<xsize; x++)
      picture[y][x] = julia(x,y);
```

```
  }

  outfile=fopen(buff,"w");
  fprintf(outfile,"P2\n%d %d %d\n",xsize, ysize, maxIteration);
  for(y=0; y<ysize; y++)
    for (x=0; x<xsize; x++)
      fprintf(outfile,"%d\n", picture[y][x]);

  fclose(outfile);
}
```

Note: The file produced is a .pgm picture file, which you can open with programs like GIMP. (`sudo apt install gimp` in Ubuntu to install it.)

Now, convert it to an MPI program which distributes the calculations in parallel. To do that, you will need to:

- Isolate the parts that talk to the user (both input and output) and restrict those to be done only by process 0.
- Broadcast the information received from the user that other processes need to know to compute their part to all the other processes.
- Split up the calculation task. I suggest that you split it on the y axis and have each processor compute a whole array in the x axis. Each process should only allocate the memory for the rows they are calculating, so as not to waste memory (except process 0 will need to allocate memory for the rows it receives as well.)
- Send all the x arrays from the other processes back to process 0. I suggest you use y as the tag so that you know which row you are receiving.
- Process 0 should print the results to the output file.