**Parallel Mandelbrot Set Image Processing**

**Purpose and content**

The purpose of this paper is to introduce the reader to the very useful subject of fractals. We will focus on the Mandelbrot set image processing. We will also show the way of visualizing the set and how to make a computer program that renders it and process an image as an output

**Introduction**

The **Mandelbrot set** is a famous example of a [fractal](https://simple.wikipedia.org/wiki/Fractal) in [mathematics](https://simple.wikipedia.org/wiki/Mathematics). It is named after [Benoît Mandelbrot](https://simple.wikipedia.org/wiki/Beno%C3%AEt_Mandelbrot" \o "Benoît Mandelbrot), a Polish-French-American [mathematician](https://simple.wikipedia.org/wiki/Mathematician). The Mandelbrot set can be explained with the [equation](https://simple.wikipedia.org/wiki/Equation) :

*zn*+1 = *zn*2 + *c*.

In that equation, *c* and *z* are [complex numbers](https://simple.wikipedia.org/wiki/Complex_number) and *n* is zero or a [positive](https://simple.wikipedia.org/wiki/Positive_number) [integer](https://simple.wikipedia.org/wiki/Integer)([natural number](https://simple.wikipedia.org/wiki/Natural_number)). Starting with *z*0=0, c is in the Mandelbrot set if the [absolute value](https://simple.wikipedia.org/wiki/Absolute_value) of *zn* never becomes larger than a certain number (that number depends on c), no matter how large *n* gets.  The equation *zn*+1 = *zn*2 + *c* was known long before Benoit Mandelbrot used a computer to visualize it. Actually, Mandelbrot was one of the first to use [computer](https://simple.wikipedia.org/wiki/Computer) [graphics](https://simple.wikipedia.org/wiki/Graphic) to create and display fractal [geometric](https://simple.wikipedia.org/wiki/Geometry) images, leading to his discovering the Mandelbrot set in 1979.

**Problem Statement**

We were given a program (mandel.c) by the course instructor written in C programming language which is the implementation of Mandelbrot Set Image Processing and it was a single threaded simple program. The Program came with a header file “bitmap.h” and also another program for creating a bitmap image which was also written in C programming language and the file name is bitmap.c. Instructions were given on how to get a bitmap image as output by running the program. In the file mandel.c the equations for Mandelbrot set were:

Xn+1 = Xn2 – Yn2 + X0

Yn+1 = 2\* Xn \* Yn + Y0

Our objective was to convert the given single threaded program to multithreaded program in order to process the output image faster when it comes to larger resolutions. It should be faster because multithreaded program will assume the image in four parts in direction to width (according to our program) and each thread will work on different parts of the image in parallel. the given program was for only black and white color. So, it was needed to add color to the output image.

**Implementation**

In order to convert the given program into multithreaded program, we declared a structure first. The structure was to put the parameters for computing the image in accordance to different threads. Because for multiple threads, we will be needing multiple sets of these parameters.

**Code:**

struct params

{

double xmin, xmax, ymin, ymax;

int smax, swidth, sheight;

int width\_start, width\_end; //for calculating in different threads

struct bitmap \*bm;

};

Then we passed the parameters altogether as an argument in a thread.

**Code:**

void compute\_image( struct bitmap \*bm, double xmin, double xmax, double ymin, double ymax, int max )

{

pthread\_t th1, th2, th3, th4;

struct params p1, p2, p3, p4;

int width = bitmap\_width(bm);

int height = bitmap\_height(bm);

p1.swidth = width;

p2.swidth = width;

p3.swidth = width;

p4.swidth = width;

p1.sheight = height;

p2.sheight = height;

p3.sheight = height;

p4.sheight = height;

p1.width\_start = 0;

p2.width\_start = width/4;

p3.width\_start = width/2;

p4.width\_start = 3\*(width)/4;

p1.xmin = xmin;

p1.xmax = xmax;

p1.ymin = ymin;

p1.ymax = ymax;

p1.smax = max;

p1.bm = bm;

p1.width\_end = width/4;

p2.xmin = xmin;

p2.xmax = xmax;

p2.ymin = ymin;

p2.ymax = ymax;

p2.smax = max;

p2.bm = bm;

p2.width\_end= width/2;

p3.xmin = xmin;

p3.xmax = xmax;

p3.ymin = ymin;

p3.ymax = ymax;

p3.smax = max;

p3.bm = bm;

p3.width\_end = 3\*(width)/4;

p4.xmin = xmin;

p4.xmax = xmax;

p4.ymin = ymin;

p4.ymax = ymax;

p4.smax = max;

p4.bm = bm;

p4.width\_end = width;

printf("creating thread \n");

pthread\_create(&th1, NULL, mandelthread, &p1);

pthread\_create(&th2, NULL, mandelthread, &p2);

pthread\_create(&th3, NULL, mandelthread, &p3);

pthread\_create(&th4, NULL, mandelthread, &p4);

pthread\_join(th1, NULL);

pthread\_join(th2, NULL);

pthread\_join(th3, NULL);

pthread\_join(th4, NULL);

}

Then we created a Thread Function.

**Code:**

void \*mandelthread(void \*arg)

{

int i,j;

struct params \*c = (struct params\*)arg;

// For every pixel in the image...

for(j=c->width\_start; j<c->width\_end;j++)

{

for(i=0;i<c->swidth;i++)

{

// Determine the point in x,y space for that pixel.

double x = c->xmin + i\*(c->xmax - c->xmin)/c->swidth;

double y = c->ymin + j\*(c->ymax - c->ymin)/c->sheight;

// Compute the iterations at that point.

int iters = iterations\_at\_point(x,y,c->smax);

// Set the pixel in the bitmap.

bitmap\_set(c->bm,i,j,iters);

}

}

}

After converting the program into multithreaded program we edited the part of the program which was for coloring the image. From bitmap.h we get that:

#ifndef GET\_RED

/\*\* Extract an 8-bit red value from a 32-bit RGBA value. \*/

#define GET\_RED(rgba) (( (rgba)>>16 ) & 0xff )

#endif

#ifndef GET\_GREEN

/\*\* Extract an 8-bit green value from a 32-bit RGBA value. \*/

#define GET\_GREEN(rgba) (( (rgba)>>8 ) & 0xff )

#endif

#ifndef GET\_BLUE

/\*\* Extract an 8-bit blue value from a 32-bit RGBA value. \*/

#define GET\_BLUE(rgba) (( (rgba)>>0 ) & 0xff )

#endif

Using these instruction, we added the following statements into our code:

int iteration\_to\_color( int i, int max )

{

int color1, color2 , color3;

if (i < 255) {

color1 = 16\*i;

color2 = 8\*(255 - i)/max;

color3 = 2;

} else if (i < 128 ) {

color1 = 16\*(128 - i)/max;

color2 = 8\*i;

color3 = 4;

} else if (i < 64) {

color1 = 16\*(i - 64);

color2 = 8\*(64- i)/max;

color3 = 6;

} else {

color1 = i\*16;

color2 = i\*8;

color3 = 3;

}

return MAKE\_RGBA(color1, color2, color3, 0);

}

**Result**

In the single threaded program we used the following commands:

Image Height: 4500

Image Width: 4500

The maximum number of iterations per point: 25000

X coordinate for image center point: 0.286932

Y coordinate for image center point: 0.014287

Scale of the image in Mandlebrot coordinates: 0.0005

The time taken to generate the image after compiling the program was: 2 minute and 34.29 seconds.

We used the same commands in the multithreaded program and the time taken to generate the image was 2 minutes and 9.54 seconds.

So, it is clear that converting the single threaded program into multithreaded program has been able to generate the image faster.

The final image that was generated: