Mandelbrot 7

Simon Yllmark

February 28, 2024

1 Introduction

In this assignment I implemented a set of module to produce a Mandelbrot image. the end result can be viewed in figure 1 and everything in between is a description of how the end result was produced.

To produce the end result 5 different modules are produced, they handles the procedures of defining the width, height and depth of the image, transforming the image (x,y) coordinates to the complex plane, calculating if the new coordinate are in Mandelbrot set for a given range value m, differentiating which coordinates that are in the set and which are not by color and finally producing the image in ppm format which was converted to png format and that can be viewed in figure 1.

1.1 Complex numbers

This module help the program transforming (x,y) coordinate to an equivalent coordinate in the complex plane. It also offer the program operation like adding two complex numbers, squaring a complex number and taking the absolute value of the complex number. The code can be viewed below:

```
    def new(r, i) do
  {r,i}
end
def add(a, b) do
  {r1,i1} = a
  {r2,i2} = b
  {r1+r2,i1+i2}
end
def sqr(a) do
  {r,i} = a
  {r*r - i*i, 2*r*i}
end
  def abs1(a) do
  {r,i} = a
```

```
    :math.sqrt((:math.pow(r,2) + :math.pow(i,2)))
  end
```

1.2 The Brot module

This module help the program to check for a given range m that a complex number c is in the Mandelbrot set. It does that by calculating if $|Z_i| > 2$ where Zi is:

- $|Z_i| = |Z_i^2 + c|$ and $Z_0 = 0$

If the test function in the module returns 0 then the c is in the set for a given range m but if i is returned instead then we can say with certainty that the complex number c is not in the set. The reason to why c is in the set for range m is that one can not guarantee that c is in the set for range $m + 1$. The code can be viewed below:

```
    def test(i,z,c,i) do
    0
  end
  def test(i,z,c,m) do
    if (Cmplx.abs1(z) > 2) do
      i
    else
      test(i + 1,calc(z,c),c,m)
    end
  end
```

1.3 The printer

The PPM module for generating an image is given.

1.4 Colors

This module colors the image that can be viewed in figure 1. The Brot module will produce two possible results where one is the integer 0 and the other is the variable i. If the value 0 is inputted as the depth parameter then the convert function will return tuple RGB number setup for the color black, meaning that all the coordinates that are in the Mandelbrot set are colored black. The code can be viewed below:

```
  def convert(depth, max) do
    a = depth * 4 / max
    x = trunc(a)
    y = trunc(255 * (a - x))
    case x do
```

```
      0 -> {:rgb, y, 0, 0}
      1 -> {:rgb, 255, y, 0}
      2 -> {:rgb, 255 - y, 255, 0}
      3 -> {:rgb, 0, 255, y}
      4 -> {:rgb, 0, 255 - y, 255}
    end
  end
```

1.5 Computing the set

This module combines the other modules into a set of steps which end result is an image. What the individual modules does have been explained in the other sections but this module also transform the (x,y) coordinate of an image into a coordinate on the complex plane that will later be calculated on. This module will then produce a list containing lists of tuples which then the PPM module use to produce the image. The code that creates an RGB color set for each pixel row by row can be viewed below:

```
def rows(_width, 0, _trans, _depth, result) do
  result
end
def rows(width, height, trans, depth, result) do
  row = Enum.map(0..width, fn(x) -> calC(x,height,trans,depth) end)
  rows(width, height - 1, trans, depth, [row | result])
end

def calC(x,height,trans,depth) do
  trans.(x,height)|>
  Brot.mandelbrot(depth)|>
  Color.convert(depth)
end
```

2 Carrying on

The end result can be viewed in figure 1, could the end result have been produced quicker. When producing this Image it took the computer a couple of second to actually produce, so there is a lot of millisecond that could have been optimised.

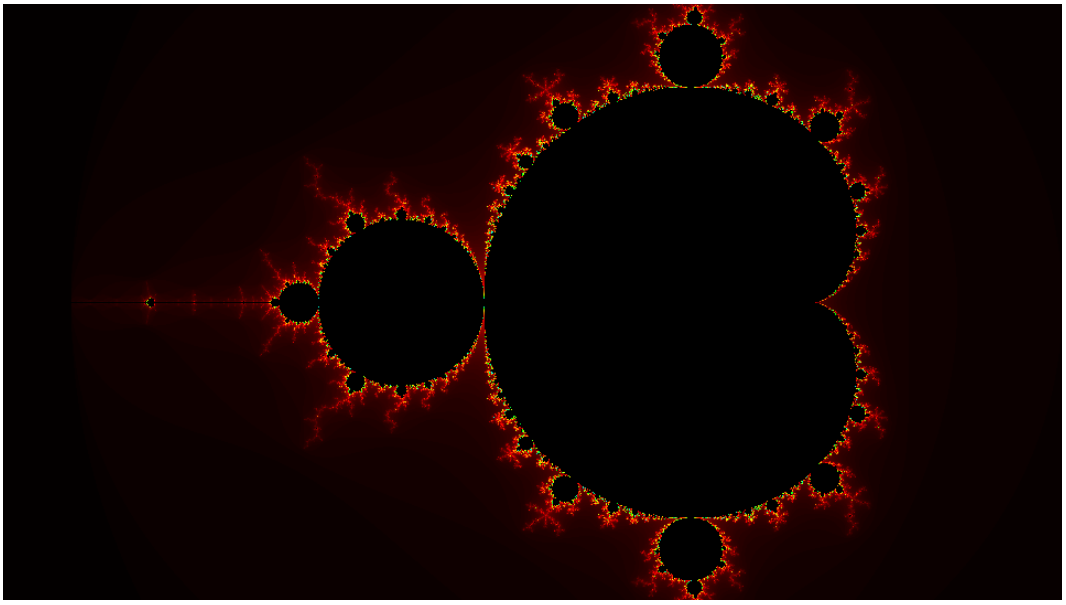

Figure 1: Mandelbrot

2.1 Discussion

The teacher Johan Montelious mentioned the use of parallelisation, that multiple processes can be executed at the same time. Considering that there is no inherent dependency between the different pixel then the work could be distributed amongst different cores in a processor.

Then he also mentioned things to consider like overhead time for kicking off a process even if the overhead time is not a lot and that there is a golden ratio between the amount of cores the processor have and the amount of processes spawned depending on the given task off course. Thus, just executing everything in parallel seem not to be the answer in this case.

3 Conclusion

The Mandelbrot set calculations in the Brot module could be improved by parallelisation because Johan gave an example with a Fibonacci sequence and the Mandelbrot calculations seem similar enough. The idea could be implemented as a separate process calculates the Z_i for all the values of i between $1 < i < 10$ and another for $10 < i < 20$ and so on. Then there could be 8 different processes being calculated on 8 different cores and if any of them return 0 then that get inputted into the color module, or i then the lowest value of i returned get inputted into the color module.