

# Kahn

Christophe Saad

April 13, 2020

## 1 Sequential

The implementation of the sequential part is mainly taken from the paper A poor man's concurrency monad. We consider the process as a monad transformer which takes an action (which contains the actual computation) and link it a continuation. Its type is

```
type 'a process = ('a -> action) -> action
```

The type action is

```
type action = Atom of (unit -> action) | Fork of action * action | Stop
```

We use **Fork** to instantiate new processes and **Atom** to represent a computation. In the **run** function, we recreate a pipeline containing the first **Fork** action to execute. Each time a **Fork** is executed, we push the two actions in the pipeline. When an **Atom** is read, we execute its computation and store his continuation back in the pipeline. This procedure ends when all the continuations are **Stop**.

## 2 Mandelbrot

This application consists of the computation and visualization of the Mandelbrot set using the implemented kahn network.

### 2.1 Mandelbrot set

The Mandelbrot set  $\mathcal{M}$  is the set of complex numbers  $c$  such that the point  $z = 0$  of the polynomial  $P_c(z) = z^2 + c$  has an orbit which remains bounded. In other words, this set can be seen as the set of complex numbers  $c$  for which the application  $z_{n+1} = z_n^2 + c$  starting from 0 for  $n \rightarrow \infty$  remains bounded.

### 2.2 Properties

A very useful property of the mandelbrot set which will make its computation easier is the following:

$$c \in \mathcal{M} \iff |\mathcal{P}_c^n(0)| \leq 2 \quad \forall n \geq 0$$

The sequence is certainly divergent if the modulus of  $z$  crosses 2.

## 2.3 Implementation

We give here a quick description of our implementation.

### 2.3.1 Objective

Our objective is to compute and plot the Mandelbrot set  $\mathcal{M}$  on a given window and range of complex numbers.

### 2.3.2 Evaluation of points

We map each pixel of the graph to a complex value depending on a given origin and a zoom scale (radius around the origin).

A common method for evaluating a point and checking whether it belongs to  $\mathcal{M}$  consists of iterating over the sequence described above starting from 0. The algorithm stops when the maximum number of steps (given as input) is reached or if we find that  $|z| > 2$  (from previous property). An obvious optimization technique would be testing  $|z|^2 > 4$  in order to avoid the squared root operation of the modulus.

### 2.3.3 Colors

A naive implementation would be to use two colors, one for pixels which belongs to  $\mathcal{M}$  and another for those that don't. Another implementation technique to visualize similarity in points would be to use the number of iterations needed for the evaluation loop to exit. We can then visualize the deep points (those which needed the maximum number of steps) and the border points.

In our algorithm, we use a technique to visualize the equipotential lines of  $\mathcal{M}$ . We replace the threshold 2 by another value  $r$  which can be seen as the escape radius. We use as inputs  $r$  and the number of iterations  $n$  to compute the potential

$$V(c) = \frac{\log(\max(1, |z_n|))}{2^n}$$

This technique stays coherent in the sense that  $V(c) = 0$  if and only if  $c \in \mathcal{M}$ . More information about the potential and required adjustments [here](#).

### 2.3.4 Kahn processes

We divide the output window into sections, each one will be handled by a process. We use one last process for plotting the resulting values of the processes computations.

Computing processes send points values (coordinates and color assigned) to the channel where they wait to be retrieved by the plotting process.

## 2.4 Usage

### 2.4.1 Running mandelbrot

The mandelbrot application can be ran with arguments (**arg** **default** **type**)

```
-w 1300 int Width of the window
-h 1000 int Height of the window
-n 1000 int Number of iterations
-p 1 int Number of processes for computation (must divide width)
-xo -0.5 float Real part of origin
-yo 0. float Imaginary part of origin
-z 1. float Zoom value (radius around origin)
-r 4. float Escape radius value
```

### 2.4.2 Some views

```
-xo -0.7463 -yo 0.1102 -z 0.005
-xo -0.7453 -yo 0.1127 -z 0.00065
-xo -0.16 -yo 1.0405 -z 0.026
-xo -0.925 -yo 0.266 -z 0.032
-xo -0.748 -yo 0.1 -z 0.0014
-xo -0.722 -yo 0.246 -z 0.019
-xo -0.235125 -yo 0.827215 -z 0.00004
-xo -0.81153120295763 -yo 0.20142958206181 -z 0.0003
```