CI4251 - Programación Funcional Avanzada
Tarea 4

Stefano De Colli
09-10203

Junio 18, 2015

Usaremos un tipo para simplificar la abstracción

```
type Complex = (Double, Double)
```

Dado que solo nos interesa la región de -2 a 2, escalamos la entrada a esos valores, independientemente de la proporción del aspecto.

```
scaleVal :: Double → Double → Double
scaleVal a v = (a - v') *2 /v'
       where
         v' = v /2

scale :: Complex → Complex → Complex
scale f (a, b) = (a', b')
     where
       (x, y) = f
       a' = scaleVal a x
       b' = scaleVal b y

toComplex :: Integral a ⇒ Complex → (a, a) → Complex
toComplex c (x, y) = scale c (f x, f y)
  where
    f = fromIntegral
```

Para converger, usaremos una función recursiva de cola para optimizar la ejecución. Además cada nos aprovechamos de los BangPatterns.

```
converge :: Complex → Word8
converge x = convergeIt 0 (0, 0) x

convergeIt :: Word8 → Complex → Complex → Word8
convergeIt 255 _ _         = 255
convergeIt !n (a, b) (x, y) = if a*a +b*b >4 then
                               n
                              else
                               convergeIt (n +1) (nr, ni) (x, y)
                              where
                               nr = a*a - b*b +x
                               ni = 2*a*b +y
```

Esta función crea una 'matriz' donde cada posición representa un pixel.

```
makeList :: (Enum b, Enum a, Num b, Num a) ⇒ a → b → [[(a, b)]]
makeList a b = foldl' (λl e → (zip (repeat e) ys):l) [] xs
  where
    xs = [0..a]
    ys = [0..b]
```

La estrategia que se utilizó fue a cada lista, evaluarla por completo con un sólo spark, y para encontrar cada valor interno, generar un spark

```
mandelStrat :: Word32 → Word32 → [[Word8]]
mandelStrat w h = S.parMap rdeepseq (S.parMap rpar converge) mv
  where
    l = makeList w h
    mv = S.parMap rdeepseq (S.parMap rpar f) l
    f = toComplex (fromIntegral w, fromIntegral h)
```

En el caso del monad paralelo, se uso una idea similar, al principio se usa un hilo para calcular las listas, y luego se mapea paralelamente esas listas, y se le aplica la otro mapeo paralelo que calcula el valor final.

```
mandelPar  :: Word32 → Word32 → [[Word8]]
mandelPar w h = runPar $ do
  fl ← spawn∘return $ makeList w h
  l ← get fl
  P.parMapM (P.parMap calculate) l
  where
    calculate c = converge $ toComplex (fromIntegral w, fromIntegral h) c
```

En el caso de REPA, nos aprovechamos de los arreglos Delayed, y una vez definido el arreglo, lo llenamos de una vez con los valores. Esta respuesta hubiese sido la más rápida si no se tuviese que picar en listas de listas, ya que chunksOf es costosa.

```
mandelREPA :: Word32 → Word32 → [[Word8]]
mandelREPA w h = chunksOf (fromIntegral w) $ R.toList d
  where
    d = R.fromFunction shape fm
    fm (Z :. x :. y) = let
                         v = toComplex (fromIntegral w, fromIntegral h) (x, y)
                       in
                         converge v
    shape = (Z :. ((fromIntegral w)::Int) :. ((fromIntegral h)::Int))
```

Para dibujar los puntos, agarramos el arreglo de arreglos y las mismas dimensiones y se mapean para dibujarlos como . con el color resultante de converge.

```
drawMandel :: Int → Int → [[Word8]] → IO ()
drawMandel w h colors = do
  G.runGraphics $ do
    window ←G.openWindow "Mandelbrot" (w, h)
    G.drawInWindow window $ G.overGraphics $
      let
        coords = makeList w h
        mk = P.zipWith (,) (P.concat coords) (P.concat colors)
      in
        P.map (λ(xy, c) → G.withTextColor (G.RGB c c c) (G.text xy ".")) mk
```

El programa de Criterion usado para correr las pruebas.

```
main :: IO ()
main = defaultMain [
         bgroup "mandelbrot" [
           bench "strat" $ nf (mandelStrat 1280) 1024
           ,bench "par" $ nf (mandelPar 1280) 1024
           ,bench "repadelay" $ nf (mandelREPA 1280) 1024
         ]
       ]
```

Resultados de 1280x1024

	Time	Mean	STD Dev	Var
Strat	1.429 s	1.445 s	16.26 ms	19 % (moderately inflated)
Par	2.168 s	2.167 s	44.17 ms	19 % (moderately inflated)
REPA D	2.068 s	1.991 s	64.47 ms	19 % (moderately inflated)