1 P_{seq}

```
type Image = V.Vector (V.Vector Int)
type Hist a = Map Int a

hbalance :: Image -> Image
hbalance img =
  let h = hist img
      a = accu h
      a0 = M.first a     -- M.function refers to functions
      agmax = M.last a   -- of the tree-based Map datastructure
      n = normalize a0 agmax a
      s = scale gmax n
      img' = apply s img
  in img'

hist :: Image -> Hist Int -- whereas V.function refers to dense-array Vector datastructure
hist = V.foldr (\i -> M.insertWith (+) i 1) M.empty . V.concat

accu :: Hist Int -> Hist Int
accu = M.scanl (+) 0

normalize :: Int -> Int -> Hist Int -> Hist Double
normalize a0' agmax' as =
    let a0 = fromIntegral a0'
        agmax = fromIntegral agmax'
        divisor = agmax - a0
    in  M.map (\freq' -> (fromIntegral freq' - a0) / divisor) as

scale :: Int -> Hist Double -> Hist Int
scale gmax = M.map (\d -> floor (d * fromIntegral gmax))

apply :: Hist Int -> Image -> Image
apply as img = V.map (V.map (M.lookupLessEqual as)) img
```

2 Work and Depth Table

- n sei die Anzahl der Bildpixel
- w sei die Bildbreite
- h sei die Bildhöhe
- p sei die Anzahl der PUs (gang members).

Table 1: Work and Depth complexities

function or variable	O(W) and O(D)
hbalance	max(n * log gmax, gmax)
hist	n * log gmax
V.concat	n
M.insertWith	log gmax
V.foldr	n * log gmax
accu	gmax
M.scanl	gmax
normalize	gmax
scale	gmax
M.map	gmax
apply	n * log gmax = w * h * log gmax
M.lookupLessEqual	log gmax
M.empty	1
M.first	1
M.last	1
V.map f xs	size(xs)*W(f,x)

3 Other aspects **e.g. sync-points, programmer workload, simplicity**

- optimisations: sequenial stream fusion is applied to the array operations here automatically. However, the program is entirely sequential
- progammer-workload: written in less than 1 hour
- simplicity: straightforward implementation from a book