1 P_{man}

```
type Image = V.Vector Int -- unboxed vector. aka dense heap array
type Hist  = V.Vector Int -- the index is the gray-value. its value the result

hbalance :: Image -> Image
hbalance img =
  let hist = parAccuHist img
      min = hist ! 0
      max = hist ! gmax
      apply hist = parMap (\i -> h ! i) img
      sclNrm :: Int -> Int
      sclNrm x = round( (x-min)/(max - min)*gmax )
  in  apply (parMap sclNrm) hist

parAccuHist :: Image -> Hist
parAccuHist [] = replicate gmax 0         -- creates [0,...,0]
parAccuHist [x] = generate gmax (\i -> if (i >= x) then 1 else 0 ) -- create [0,...0,1,...,1]
parAccuHist xs =
  let (left,right) = splitMid xs
      [leftRes,rightRes] = parMap parAccuHist [left,right] -- two parallel recursive calls
  in  parZipWith (+) leftRes rightRes
```

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)	O(D)
hbalance	n * gmax	log n
apply	n	1
parMap sclNrm	gmax	1
parAccuHist	n * gmax	log n
parAccuHist	n * gmax	log n
splitMid	1	1
parZipWith	gmax	1
replicate	gmax	1
generate	gmax	1
parMap f xs	W(f,x)*size(xs)	1
arr ! i	1	1

3 Calculating "hbalance"

$$\begin{aligned}
\mathrm{W}(n, gmax) &= \mathrm{W}(parAccuHist) + \mathrm{W}(parMap, sclNrm) + \mathrm{W}(apply) \\
&= n \cdot gmax + gmax + n \\
&\in O(n \cdot gmax) \\
\mathrm{D}(n, gmax) &= \max\{parAccuHist, (parMap, sclNrm), apply\} \\
&= \max\{\log n, 1, 1\} \\
&\in O(\log n)
\end{aligned} \tag{1}$$

4 Calculating "parAccuHist"

On each recursive call of parAccuHist, there are consantly many calls to functions of work O(gmax) and depth O(1). The only exception are the two recursive calls. They call the problem with array of half-size. The depth of this function is logarithmic to the input size, because of this type of recursive calls. (If gmax is treated as a constant, then the following result can also be derived from the Master Theorems first case.)

$$
\begin{aligned}
\mathrm{W}(n, gmax) &= \begin{cases} gmax & \text{if } n \leq 1 \\ 2\mathrm{W}(\frac{n}{2}) + gmax & \text{else} \end{cases} \\
&= \begin{cases} gmax & \text{if } n \leq 1 \\ gmax2^0 + 2^1\mathrm{W}(\frac{n}{2}) & \text{if } n = 2 \\ gmax2^0 + gmax2^1 + 2^2\mathrm{W}(\frac{n}{4}) & \text{if } n = 3 \\ gmax2^0 + gmax2^1 + ... + gmax2^{\log n - 1} + 2^{\log n}\mathrm{W}(1) & \text{else} \end{cases} \\
&= \text{ (... tying the knot ...)} \\
&= gmax \sum_{i=0}^{\log n} 2^i \\
&= gmax(2^{\log n + 1} - 1) \\
&= gmax(2n - 1) \\
&\in O(n \cdot gmax) \\
\mathrm{D}(n, gmax) &\in O(\log n)
\end{aligned}
\tag{2}
$$

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

- optimisations: ?
- progammer-workload: ?
- simplicity: ?