1 The algorithm

```
type Many a = [: a :]
type Image a = [:[: a :]:]

type Hist a = [: a :]

hbalanceBulk :: Many (Image Int) -> Many (Image Int)
hbalanceBulk = mapP hbalance

hbalance :: Image Int -> Image Int
hbalance img =
    let h = hist img
        a = accu h
        a0 = headP a
        agmax = lastP a
        n = normalize a0 agmax a
        s = scale gmax n
        img' = apply s img
    in img'

gmax :: Int
gmax = 255

hist :: Image Int -> Hist Int
hist =
    sparseToDenseP (gmax+1) 0
    . mapP (\g -> (headP g,lengthP g))
    . groupP
    . sortP
    . concatP

accu :: Hist Int -> Hist Int
accu = scanlP (+) 0

normalize :: Int -> Int -> Hist Int -> Hist Double
normalize a0' agmax' as =
    let a0 = P.fromIntegral a0'
        agmax = P.fromIntegral agmax'
        divisor = agmax D.- a0
    in [: (P.fromIntegral freq' D.- a0) D./ divisor | freq' <- as :]

scale :: Int -> Hist Double -> Hist Int
scale gmax as = [: P.floor (a D.* P.fromIntegral gmax) | a <- as :]

apply :: Hist Int -> Image Int -> Image Int
apply as img = mapP (mapP (as !:)) img
```

2 Utilised Functions

Table 1: Utilised function with their type signatures and their work and depth complexity.

Function	Work	Depth	Type
hbalanceBulkd imgs	?	?	`[:[:[:Int:]:]:] -> [:[:[:Int:]:]:]`
hbalance img	?	?	`[:[:Int:]:] -> [:[:Int:]:]`
hist img	$1 + wh(2 + \log(wh)) + gmax$	$\log gmax$	`[:[:Int:]:] -> [:Int:]`
accu h	$6n - 1$	$\log n$	`[:Int:] -> [:Int:]`
scale gmax n	$2n + 1$	1	`Int -> [:Double:] -> [:Int:]`
normalize a_0 a_{gmax} a	$2n - 1$	1	`Int -> Int -> [:Int:] -> [:Double:]`
apply s img	$1 + w + wh$	1	`[:Int:] -> [:[:Int:]:] -> [:[:Int:]:]`
mapP f xs	n	1	`(a -> b) -> [:[:a:]:] -> [:[:b:]:]`
headP xs	1	1	`[:[:a:]:] -> a`
lastP xs	1	1	`[:[:a:]:] -> a`
concatP xss	1	1	`[:[:a:]:] -> [:a:]`
sortP xs	$n \log n$	$\log n$	`Ord a => [:a:] -> [:a:]`
groupP xs	$(n + \text{groups in xs}) \log n$	$\log n$	`Eq a => [:a:] -> [:[:a:]:]`
sparseToDenseP xs	$k \cdot L(\text{result})$	1	`Int -> Int -> [:(a,Int):] -> [:a:]`
lengthP xs	1	1	`[:a:] -> Int`
scanlP f z xs	$6n - 1$	$\log n$	`(a -> a -> a) -> a -> [:a:] -> [:a:]`
indexP,!: i xs	1	1	`[:a:] -> Int -> a`

where p is the number of processors. Functions in *italics* denote an $O(\cdot)$-class rather than the actual number of steps. Note, that various array class constraints (PAElt a =>...) on array functions ommitted were omitted.

3 Variation of the number of processors

With P processors and a cross-process communication latency of L, a program with work W and depth D will run time T with such that the following holds:

$$\frac{W}{P} \leq T \leq \frac{W}{P} + L \cdot D$$

Note that, W and D have to be the exact numbers rather than an $O(\cdot)$-class

4 scanlP

$$\begin{aligned} \mathrm{W}(n) &= n(2 + \mathrm{W}(f)) + \mathrm{W}\left(\frac{n}{2}\right) \\ &= n(2 + \mathrm{W}(f))(2 - 2^{\log_2 n}) + 1 \\ &= (2 + \mathrm{W}(f))(2n - 1) + 1 \\ &= 6n - 1 \\ \mathrm{D}(n) &= \log n \end{aligned} \tag{1}$$

5 sparseToDenseP

```
sparseToDenseP :: Enum e => e -> a -> [: (e,a) :] -> [: a :]
sparseToDenseP              many init map             result
sparseToDenseP 0 7 9 [:   (1,5),(2,4),(6,7)   :]  == [:  0,5,4,0,0,0,7,0   :]
```

sparseToDense n z map creates an array of length n where the element at the index i has x if (i,x) is in the map, or z otherwise. In effect it turns a sparse vector to a dense one

6 groupP

$$\begin{aligned} \mathrm{W}(n) &= (n + \text{unique elements in xs}) \log n \\ \mathrm{D}(n) &= \log n \end{aligned} \tag{2}$$

7 hbalance

Let $|img| = w \cdot h$ and $n = \text{gmax}$.

$$\begin{aligned}
&\mathrm{W}(w \times h) \\
&\quad = \mathrm{W}(hist) + \mathrm{W}(accu) + \mathrm{W}(headP) + \mathrm{W}(lastP) + \mathrm{W}(normalize) + \mathrm{W}(scale) + \mathrm{W}(apply) \\
&\quad = (1 + |img|(2 + \log|img|) + n) + (6n - 1) + 1 + 1 + (2n + 1) + (2n + 1) + (1 + w + |img|) \\
&\quad = w + |img|(3 + \log(|img|)) + 11n + 5 \\
&\quad \in O\left(|img| \log|img| + gmax\right) \\
&\mathrm{D}(w \times h) \\
&\quad = 1 + \max\{\mathrm{D}(hist), \mathrm{D}(accu), \mathrm{D}(headP), \mathrm{D}(lastP), \mathrm{D}(normalize), \mathrm{D}(scale), \mathrm{D}(apply)\} \\
&\quad = 1 + \max\{\log|img|, \log gmax\}
\end{aligned} \tag{3}$$

With P processors we have an expected runtime of T where:

$$\begin{aligned}
\frac{W}{P} &\leq T \leq \frac{W}{P} + D \\
\frac{|img|\log|img| + gmax}{P} &\leq T \leq \frac{|img|\log|img| + gmax}{P} + \max\{\log|img|, \log gmax\} \\
&\text{Fall 1:} |img| > gmax. \text{Es gibt mehr Bildpixel als es zulässige Grauwerte gibt.} \\
\frac{2|img|\log|img|}{P} &\leq T \leq \frac{2|img|\log|img|}{P} + \log|img| \qquad | \div (2\log|img|) \\
\frac{|img|}{P} &\leq T \leq \frac{|img|}{P} + \frac{1}{2} \\
&\text{Fall 2:} gmax > |img|. \text{Der Grauwertbereich ist größer als die Anzahl der Pixel} \\
\frac{2gmax\log gmax}{P} &\leq T \leq \frac{2gmax\log gmax}{P} + \log gmax \qquad | \div (2\log gmax) \\
\frac{gmax}{P} &\leq T \leq \frac{gmax}{P} + \frac{1}{2}
\end{aligned} \tag{4}$$

8 hist

Let $|img| = w \cdot h$ and $n = \text{gmax}$.

$$\begin{aligned}
&\mathrm{W}(w \times h) \\
&\qquad = \mathrm{W}(concatP) + \mathrm{W}(sortP) + \mathrm{W}(groupP) + 2\mathrm{W}(mapP) + \mathrm{W}(sparseToDenseP) \\
&\qquad = 1 + |img| \log |img| + 2|img| + n \\
&\qquad = 1 + |img|(2 + \log |img|) + n \\
&\mathrm{D}(w \times h) = \log |img|
\end{aligned} \tag{5}$$

9 accu

$$\begin{aligned}
\mathrm{W}(n) &= \mathrm{W}(scanlP) \\
&= 6n - 1 \\
\mathrm{D}(n) &= \mathrm{D}(scanlP) \\
&= \log n
\end{aligned} \tag{6}$$

10 normalize

$$\begin{aligned}
\mathrm{W}(n) &= 2n + 1 \\
\mathrm{D}(n) &= 1
\end{aligned} \tag{7}$$

11 apply

Let w and h be the width and height of the image (img).

$$\begin{aligned}
\mathrm{W}(w \times h) &= 1 + w \cdot \mathrm{W}(\text{inner}) \\
&= 1 + w(1 + h) \\
&= 1 + w + wh \\
\mathrm{D}(w \times h) &= 1
\end{aligned} \tag{8}$$