1 P_{vect}

```
type Image = PA (PA Int)
type Hist = PA Int

hbalance :: Image -> Image
hbalance img =
  let a :: Hist                                  -- Histogram
      a = joinD                                  -- accu end
          . mapD (\(as,a) -> mapS (plusInt a) as)
          . propagateD plusInt 0
          . mapD (scanlS plusInt 0)              -- accu begin
          . sparseToDenseD (plusInt gmax 1) 0    -- hist end
          . splitSparseD (plusInt gmax 1)
          . joinD
          . mapD tripletToATup2
          . segdSplitMerge 0
          . sortPS
          . concatPS                             -- hist begin
          $ img                                  -- 1

      n :: Int
      n = lengthPS a                             -- 2

      a0, divisor, gmax' :: Double
      a0      = int2Double . headPS $ a          -- 3, variables for normalize and scale
      divisor = minusDouble (int2Double (lastPS a)) . int2Double . headPS $ a
      gmax'   = int2Double $ gmax

      normScale :: Int -> Int                    -- 0, body of normalize and scale
      normScale =
        floorDouble
        . (flip multDouble) gmax'
        . (flip divDouble) divisor
        . (flip minusDouble) a
        . int2Double

      as :: Hist                                 -- final mapping array
      as = joinD . mapD (mapS normScale) . splitD $ a -- 4, normalize and scale applied

      pixelReplicate :: Hist -> PA Hist          -- 0, artifact of NDP
      pixelReplicate = concatPS . replPL (lengths (getSegd xs)) . replPS (lengthPS img)

  in unconcatPS img
     . indexPL (pixelReplicate as)               -- 5, apply. core of nested data parallelism here!
     . concatPS
     $ 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)	O(D)
hbalance	$\max(n \log n, gmax)$	$\log \max(n, gmax)$
hist	$\max(n \log n, gmax)$	$\log n$
concatPS	1	1
sortPS	$n \log n$	$\log n$
segdSplitMerge	$n \log n$	$\log n$
mapD tripletToATup2	gmax	1
joinD xs	gmax	1
accu	gmax	$\log gmax$
sparseToDenseD	gmax	1
splitSparseD	gmax	1
mapD scanlS	gmax	1
propagateD	1	$\log gmax$
mapD (mapS plusInt)	gmax	1
joinD	gmax	1
as	gmax	1
splitD	gmax	1
mapD normScale	gmax	1
joinD	gmax	1
pixelReplicate	1 (bzw. $n \cdot gmax$)	1
img’	n	1
concatPS	1	1
indexPL	n	1
unconcatPS	1	1
	general complexities	
mapD fS xs	W$(fS, s) * size(xs)$	W$(fS, s) * size(xs)/p$
joinD xs	size(xs)	1 (communication overhead?)
splitD xs	size(xs)	1 (communication overhead?)
replPS n x	n	1
replPL ns xs	sum(ns)*size(xs)	1
sortPS xs	$size(xs) \log size(xs)$	$\log size(xs)$

3 Calculating "hbalance" entirely

Let $n = |img| = w \cdot h$

$$\begin{aligned}
\mathrm{W}(w \times h, gmax) &= \mathrm{W}(hist) + \mathrm{W}(accu) + \mathrm{W}(as) + \mathrm{W}(pixelReplicate) + \mathrm{W}(img') \\
&= O(\max(n \log n, gmax) + gmax + gmax + 1 + n) \\
&\in O(\max(n \log n, gmax)) \\
\mathrm{D}(w \times h, gmax) &= \max\{hist, accu, as, pixelReplicate, img'\} \\
&= \max\{\log n, \log gmax\} \\
&\in O(\log \max(n, gmax))
\end{aligned} \tag{1}$$

4 Calculating the histogram "hist"

Let $n = |img| = w \cdot h$

$$\begin{aligned}
\mathrm{W}(w \times h) &= \mathrm{W}(concatPS) + \mathrm{W}(sortPS) + \mathrm{W}(segdSplitMerge) \\
&\quad + \mathrm{W}(mapD, tripletToATup2) + \mathrm{W}(joinD) \\
&= 1 + n \log n + n \log n + gmax + gmax \\
&= 1 + 2(n \log n + gmax) \\
&\in O(\max(n \log n, gmax)) \\
\mathrm{D}(w \times h) &= \max\{concatPS, sortPS, ..., joinD\} \\
&\in O(\log n)
\end{aligned} \tag{2}$$

5 Calculating the accumulated histogram "accu"

$$\begin{aligned}
\mathrm{W}(gmax) &= \mathrm{W}(splitSparseD) + \mathrm{W}(sparseToDenseD) + \mathrm{W}(mapD, scanlS) \\
&\quad + \mathrm{W}(propagateD) + \mathrm{W}(mapD, mapS, plusInt) + \mathrm{W}(joinD) \\
&= gmax + gmax + gmax + 1 + gmax + gmax \\
&= 1 + 5 \cdot gmax \\
&\in O(gmax) \\
\mathrm{D}(gmax) &= \max\{splitSparseD, sparseToDenseD, (mapD, scanlS), ...\} \\
&\in O(\log gmax)
\end{aligned} \tag{3}$$

6 Calculating the mapping array "as"

$$\begin{aligned}
\mathrm{W}(gmax) &= \mathrm{W}(joinD) + \mathrm{W}(mapD, normScale) + \mathrm{W}(splitD) \\
&= gmax + gmax + gmax \\
&= 3 \cdot gmax \\
&\in O(gmax) \\
\mathrm{D}(gmax) &= \max\{joinD, (mapD, normScale), splitD\} \\
&\in O(1)
\end{aligned} \tag{4}$$

7 Calculating by "pixelReplicate"

PixelReplicate is a function which executed strictly and literally would have a work complexity of $n \cdot gmax$. However, due to strong improvements in [WorkEfficient] this replication is simply cached locally such that the work complexity for this case is negilible and can be simplified to 1.

$$\begin{aligned} \mathrm{W}(gmax, w \times h) \in O(1) \\ \mathrm{D}(gmax, w \times h) \in O(1) \end{aligned} \tag{5}$$

8 Calculating the final balanced image "img'"

$$\begin{aligned} \mathrm{W}(gmax, w \times h) &= \mathrm{W}(unconcatPS) + \mathrm{W}(indexPL) + \mathrm{W}(concatPS) \\ &= 1 + n + 1 \\ &\in O(n) \\ \mathrm{D}(gmax, w \times h) &= \max\{unconcatPS, indexPL, concatPS\} \\ &\in O(1) \end{aligned} \tag{6}$$

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

- optimisations: removed sync-points(more local operations), tight distributed normalisation loop, distributed optimal prefix-sum, distributed groupP and sortP, nested data parallelism enables a depth lower than O(width)
- progammer-workload: compiler does all the optimisations. as much as for P_{nest}
- simplicity: hard to comphrehend, not indented for human readers, but shows optimisations clearly