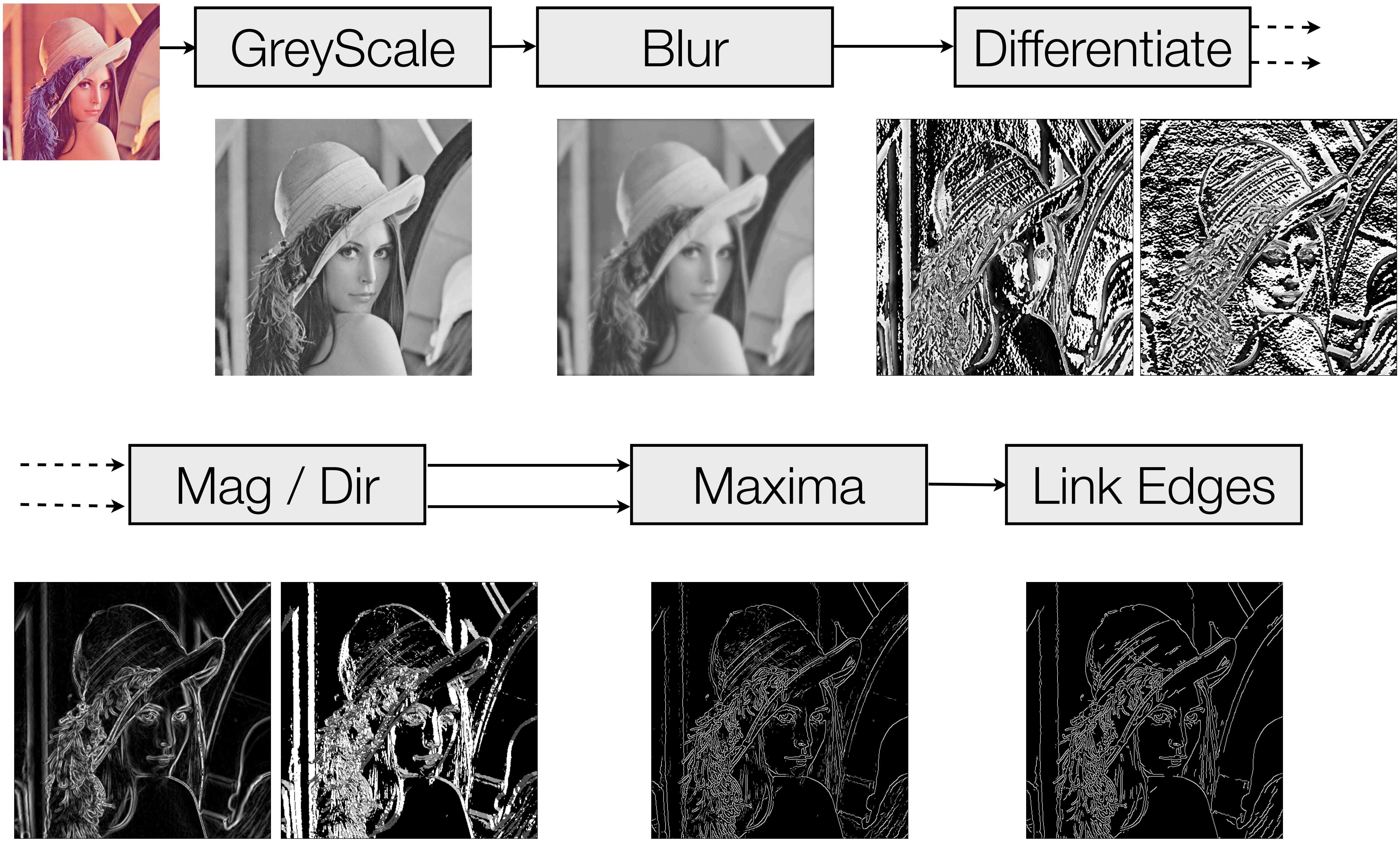
Efficient Parallel Stencil Convolution in Haskell

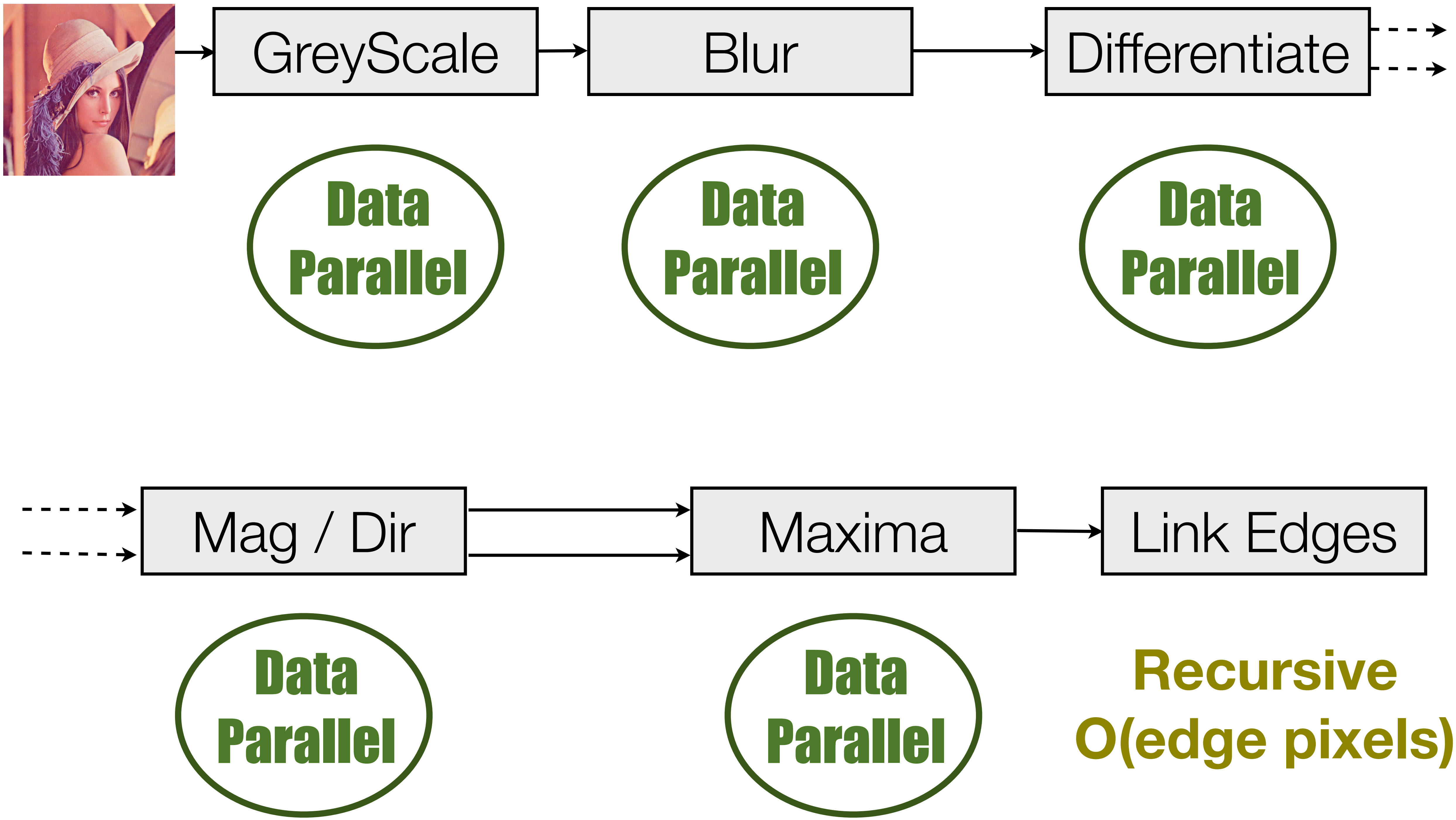
Ben Lippmeier
University of New South Wales
FP-SYD 2011/03/17

Canny Edge Detection

Canny Edge Detection



Canny Edge Detection



Canny Edge Detection

GreyScale → Blur → Differentiate

Pixel/Pixel

$$Binomial_{7X}$$
$$\begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix}$$

$$Binomial_{7Y}$$
$$\begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix}^T$$

Stencil Convolution

$$Sobel_X \quad Sobel_Y$$
$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Stencil Convolution

Mag / Dir → Maxima → Link Edges

Pixel/Pixel

Comparison of adjacent pixels

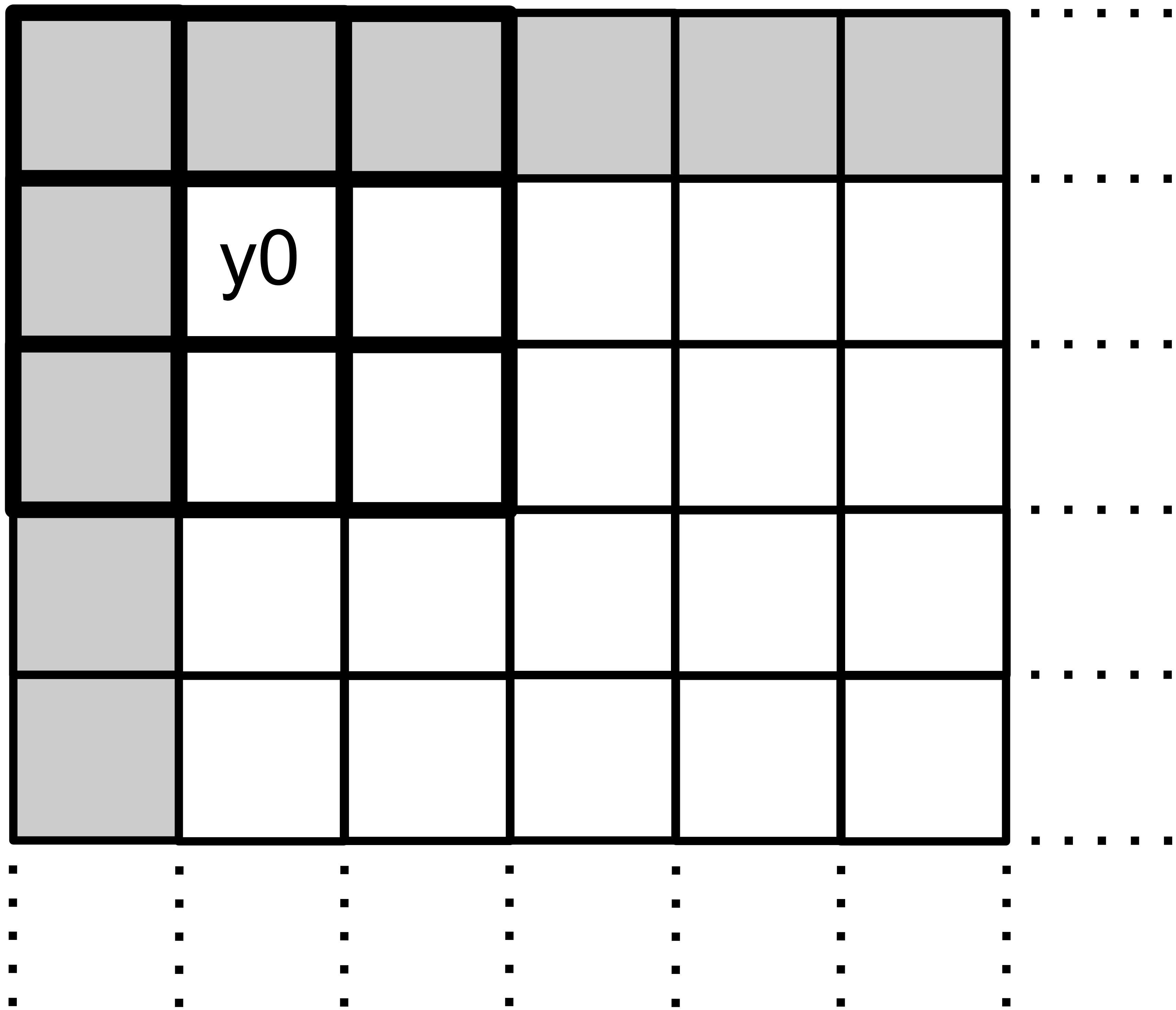
Wildfire Algorithm

A single point result from a 3x3 stencil.

$$(A * K)(x, y) = \sum_i \sum_j A(x + i, y + j)\, K(i, j)$$

```
r = a[i-1][j-1] * k[-1][-1]
  + a[i-1][j  ] * k[-1][ 0]
  + a[i-1][j+1] * k[-1][+1]

  + a[i  ][j-1] * k[ 0][-1]
  + a[i  ][j  ] * k[ 0][ 0]
  + a[i  ][j+1] * k[ 0][+1]

  + a[i+1][j-1] * k[+1][-1]
  + a[i+1][j  ] * k[+1][ 0]
  + a[i+1][j+1] * k[+1][+1]
```

Don’t. push. me. cause. I’m. close. to. the. edge....



Testing the border at every pixel is slow....

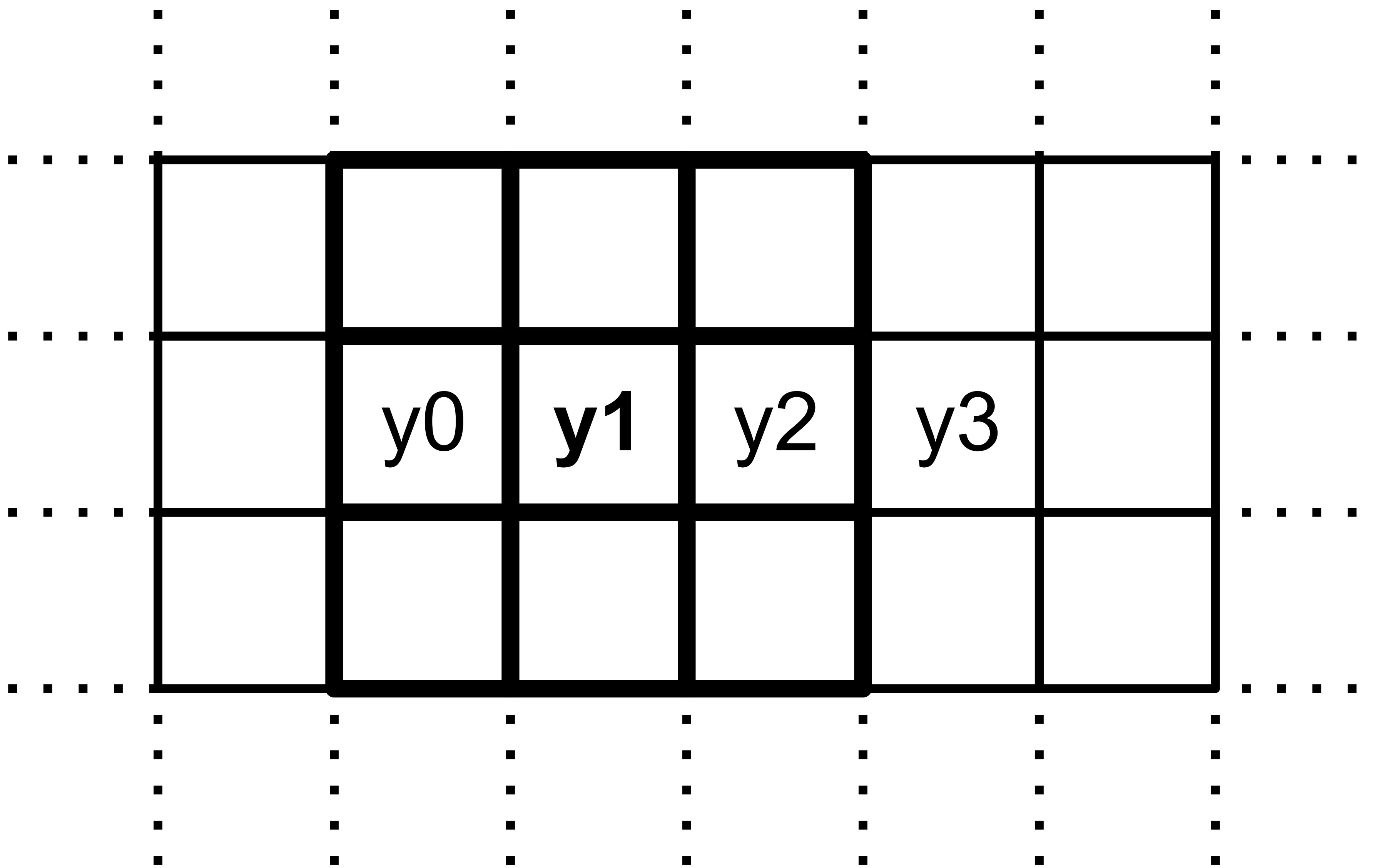
```
{-# INLINE relaxLaplace #-}
relaxLaplace :: Image -> Image
relaxLaplace arr
 = traverse arr id elemFn
 where  _ :. height :. width = extent arr

        {-# INLINE elemFn #-}
        elemFn get d@(Z :. i :. j)
         = if isBorder i j
            then  get d
            else (get (Z :. (i-1) :. j)
              +   get (Z :. i     :. (j-1))
              +   get (Z :. (i+1) :. j)
              +   get (Z :. i     :. (j+1))) / 4

        {-# INLINE isBorder #-}
        isBorder i j
         =  (i == 0) || (i >= width  - 1)
         || (j == 0) || (j >= height - 1)
```

Testing the border at every pixel is slow....

```
{-# INLINE relaxLaplace #-}
relaxLaplace :: Image -> Image
relaxLaplace arr
 = traverse arr id elemFn
 where  _ :. height :. width = extent arr

        {-# INLINE elemFn #-}
        elemFn get d@(Z :. i :. j)
         = if isBorder i j
            then  get d
            else (get (Z :. (i-1) :. j)
              +   get (Z :. i     :. (j-1))
              +   get (Z :. (i+1) :. j)
              +   get (Z :. i     :. (j+1))) / 4

        {-# INLINE isBorder #-}
        isBorder i j
         =   (i == 0) || (i >= width  - 1)
         || (j == 0) || (j >= height - 1)
```

DIE!

Sharing in computations of adjacent pixels.



3*3*4 = 36
3*6 = 18

36 / 18 = 2

Application of a single Laplace stencil.

0	1	0
1	0	1
0	1	0

```
case quotInt# ixLinear width of { iX ->
case remInt#  ixLinear width of { iY ->
 writeFloatArray# world arrDest ixLinear
  (+##  (indexFloatArray# arrBV
            (+# arrBV_start (+# (*# arrBV_width iY) iX)))
   (*## (indexFloatArray# arrBM
            (+# arrBM_start (+# (*# arrBM_width iY) iX)))
    (/## (+## (+## (+##
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# (-# width 1) iY) iX)))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# width iY) (-# iX 1)))))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# (+# width 1) iY) iX))))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# width iY) (+# iX 1)))))
     4.0)))
 }}
```

Application of a single Laplace stencil.

0	1	0
1	0	1
0	1	0

```
case quotInt# ixLinear width of { iX ->
case remInt#  ixLinear width of { iY ->
 writeFloatArray# world arrDest ixLinear
  (+##  (indexFloatArray# arrBV
          (+# arrBV_start (+# (*# arrBV width iY) iX)))
   (*## (in
          (                                          ) iX)))
    (/## (+
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# (-# width 1) iY) iX)))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# width iY) (-# iX 1)))))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# (+# width 1) iY) iX))))
          (indexFloatArray# arrSrc
           (+# arrSrc_start (+# (*# width iY) (+# iX 1)))))
     4.0)))
 }}
```

x + y * width

Two new features:

Partitioned arrays
Represent the partitioning into border and internal regions directly, to avoid the test in the inner loop.

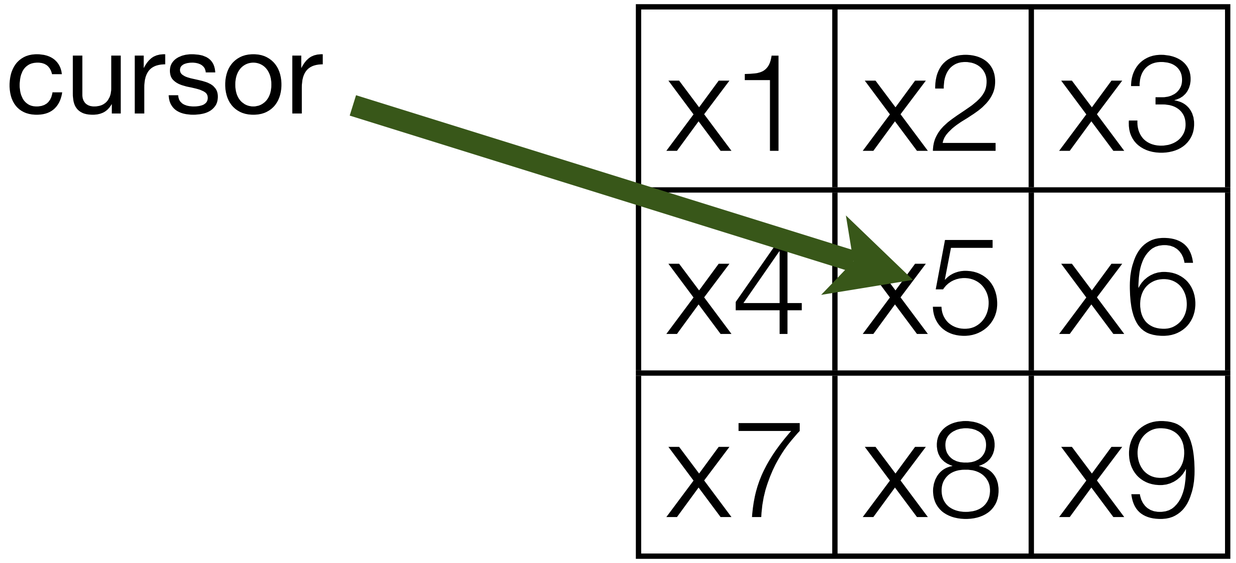
Cursored arrays
Expose intermediate linear indices when calculating array offsets, to avoid repeated use of `x + y * width`.

New Repa Array Types:

```
data Array sh a
    = Array       { arrayExtent  :: sh
                  , arrayRegions :: [Region sh a] }
data Region sh a
    = Region      { regionRange  :: Range sh
                  , regionGen    :: Generator sh a }
data Range sh
    = RangeAll
    | RangeRects  { rangeMatch   :: sh -> Bool
                  , rangeRects   :: [Rect sh] }
data Rect sh
    = Rect sh sh
```

New Repa Array Types:

```
data Generator sh a
 = GenManifest { genVector :: Vector a }

 | forall cursor.
   GenCursored { genMake  :: sh -> cursor
               , genShift :: sh -> cursor -> cursor
               , genLoad  :: cursor -> a }
```



Defining the stencil

```
data Stencil sh a
   = Stencil { stencilSize  :: sh
             , stencilZero  :: b
             , stencilAcc   :: sh -> a -> a -> a }

makeStencil :: sh -> (sh -> Maybe a) -> Stencil sh a
makeStencil ex getCoeff
        =  Stencil ex 0
        $  \ix val acc
              -> case getCoeff ix of
                   Nothing      -> acc
                   Just coeff   -> acc + val * coeff

 laplace :: Stencil sh a
 laplace =  makeStencil (Z :. 3 :. 3)
        $  \ix -> case ix of
                   Z :.  0 :.  1 -> Just 1
                   Z :.  0 :. -1 -> Just 1
                   Z :.  1 :.  0 -> Just 1
                   Z :. -1 :.  0 -> Just 1
                   _             -> Nothing
```

Defining the stencil

```
data Stencil sh a
   = Stencil { stencilSize  :: sh
             , stencilZero  :: b
             , stencilAcc   :: sh -> a -> a -> a }

makeStencil :: sh -> (sh -> Maybe a) -> Stencil sh a
makeStencil ex getCoeff
        =  Stencil ex 0
        $  \ix val acc
             -> case getCoeff ix of
                  Nothing     -> acc
                  Just coeff  -> acc + val * coeff

 laplace :: Stencil sh a
 laplace = [|stencil2 0 1 0
                      1 0 1
                      0 1 0 |]
```

Not a Number

```
{-# RULES
    “add-id” forall (x :: Float). x + 0 = x
    “mul-id” forall (x :: Float). x * 0 = 0
#-}
```

Not a Number

```
{-# RULES
     "add-id" forall (x :: Float). x + 0 = x
     "mul-id" forall (x :: Float). x * 0 = 0
#-}
```

With IEEE 754 Floats

$\infty * 0 = \text{NaN}$

Not a Number

```
{-# RULES
    "add-id" forall (x :: Float). x + 0 = x
    "mul-id" forall (x :: Float). x * 0 = 0
#-}
```

```
makeStencil :: sh -> (sh -> Maybe a) -> Stencil sh a
makeStencil ex getCoeff
       =  Stencil ex 0
       $  \ix val acc
              -> case getCoeff ix of
                   Nothing     -> acc
                   Just coeff  -> acc + val * coeff
```

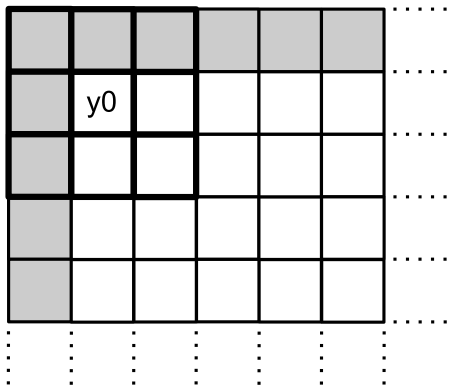
Applying a Stencil

```
-- | Compute gradient in the X direction.
gradientX :: Array DIM2 Float -> Array DIM2 Float
gradientX img
   = force2 $ forStencil2 (BoundConst 0) img
     [stencil2| -1  0  1
                -2  0  2
                -1  0  1 |]
```

Detection of Local Maxima

```
-- | Suppress pixels which are not local maxima.
maxima :: Float -> Float -> Image (Float, Float) -> Image Word8
maxima threshLow threshHigh dMagOrient
 = force2 $ makeBordered2 (extent dMagOrient) 1 (GenCursor id addDim (const 0))
                                                (GenCursor id addDim compare)
 where compare ix@(sh :. i :. j)
        | o == undef    = edge None
        | o == horiz    = isMax (getMag (sh :. i   :. j-1)) (getMag (sh :. i   :. j+1))
        | o == vert     = isMax (getMag (sh :. i-1 :. j))   (getMag (sh :. i+1 :. j))
        | o == negDiag  = isMax (getMag (sh :. i-1 :. j-1)) (getMag (sh :. i+1 :. j+1))
        | o == posDiag  = isMax (getMag (sh :. i-1 :. j+1)) (getMag (sh :. i+1 :. j-1))
        | otherwise     = edge None
       where
         o     = getOrient ix
         m     = getMag    ix

         getMag     = fst . (dMagOrient !)
         getOrient  = snd . (dMagOrient !)

         isMax mag1 mag2
           | m < threshLow   = edge None
           | m < mag1        = edge None
           | m < mag2        = edge None
           | m < threshHigh  = edge Weak
           | otherwise       = edge Strong
```

```
mapStencil2
  :: Boundary a -> Stencil DIM2 a -> Array DIM2 a -> Array DIM2 a

mapStencil2 boundary (Stencil sExtent _ _) arr
 = let (Z :. aHeight :. aWidth) = extent arr
       (Z :. sHeight :. sWidth) = sExtent

       rectsInternal    = ...
       rectsBorder      = ...
       inInternal ix    = ...
       inBorder   ix    = ...

       make  (Z:.y:.x)  = Cursor (x + y*aWidth)
       shift (Z:.y:.x) (Cursor offset)
        = Cursor (offset + x + y*aWidth)

       loadBorder ix    = case boundary of ...
       loadInner cursor = unsafeAppStencil2 stencil arr shift cursor

   in  Array (extent arr)
        [ Region (RangeRects inBorder rectsBorder)
                 (GenCursored id addIndex loadBorder)

        , Region (RangeRects inInternal rectsInternal)
                 (GenCursored make shift loadInner) ]
```



```haskell
unsafeAppStencil2
  :: Stencil DIM2 a -> Array DIM2 a
  -> (DIM2 -> Cursor -> Cursor)        -- shift cursor
  -> Cursor -> a

unsafeAppStencil2
  stencil@(Stencil sExtent sZero sAcc)
  arr@(Array aExtent [Region RangeAll (GenManifest vec)])
  shift cursor

  | _ :. sHeight :. sWidth  <- sExtent
  , sHeight <= 3, sWidth <= 3
  = template3x3 loadFromOffset sZero

  | otherwise = error "stencil too big for this method"

  where getData (Cursor index)
         = vec `unsafeIndex` index

        loadFromOffset oy ox
         = let  offset = Z :. oy :. ox
                cur'   = shift offset cursor
           in   sAcc offset (getData cur')
```

```
template3x3 :: (Int -> Int -> a -> a) -> a -> a
template3x3 f sZero
  =  f (-1) (-1)  $  f (-1)   0  $  f (-1)   1
  $  f   0  (-1)  $  f   0    0  $  f   0    1
  $  f   1  (-1)  $  f   1    0  $  f   1    1
  $  sZero
```

... dreaming of supercompilation

```haskell
fillCursoredBlock2
 :: Elt a => IOVector a             -- vec
 -> (DIM2    -> cursor)             -- makeCursor
 -> (DIM2    -> cursor -> cursor) -- shiftCursor
 -> (cursor -> a) -> Int            -- loadElem, width
 -> Int -> Int -> Int -> Int        -- x0 y0 x1 y1
 -> IO ()

fillCursoredBlock2 !vec !make !shift !load !width !x0 !y0 !x1 !y1
 = fillBlock y0
 where
  fillBlock !y
   | y > y1            = return ()
   | otherwise
   = do   fillLine4 x0
          fillBlock (y + 1)
   where
    fillLine4 !x
     | x + 4 > x1      = fillLine1 x
     | otherwise
     = do BODY
          fillLine4 (x + 4)

     fillLine1 !x
      | x > x1     = return ()
      | otherwise
      = do unsafeWrite vec (x + y * imageWidth)
                 (getElem $ makeCursor (Z:.y:.x))
           fillLine1 (x + 1)
```

```
fillLine4 !x
 | x + 4 > x1      = fillLine1 x
 | otherwise
 = do let srcCur0 = make  (Z:.y:.x)
      let srcCur1 = shift (Z:.0:.1) srcCur0
      let srcCur2 = shift (Z:.0:.1) srcCur1
      let srcCur3 = shift (Z:.0:.1) srcCur2

      let val0    = load srcCur0
      let val1    = load srcCur1
      let val2    = load srcCur2
      let val3    = load srcCur3

      let !dstCur0 = x + y * width
      unsafeWrite vec (dstCur0)     val0
      unsafeWrite vec (dstCur0 + 1) val1
      unsafeWrite vec (dstCur0 + 2) val2
      unsafeWrite vec (dstCur0 + 3) val3
      fillLine4 (x + 4)
```

```
$wa4_s3HS =
  \ (ww4_s3lq :: Int#) (w2_s3ls :: State# RealWorld) ->
    case >=# (+# ww4_s3lq 4) ipv8_i30r of _ {
      False ->
        let { a22_s4SQ = +# ww4_s3lq (*# ww3_s3ly ipv1_X2LM) } in
        let { Vector rb_i2YQ _ rb2_i2YS ~ _ <- ds6_d2b5 `cast` ... } in
        let { a23_i30Y = +# ww4_s3lq (*# ww3_s3ly ipv1_X2LM) } in
       let { __DEFAULT ~ s#_X39w
       <- writeFloatArray#
            arr#_i2Pd
            a23_i30Y
            (plusFloat#
               (plusFloat#
                  (plusFloat#
                     (plusFloat#
                        (plusFloat#
                           (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) 1)))
                           (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) (-1)))) __float -1.0))
                        (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ 1))) __float 2.0))
                     (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ (-1)))) __float -2.0))
                  (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ (*# (-1) ipv1_X2LM)) 1))))
               (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
            (w2_s3ls `cast` ...)
       } in
       let { a24_s4TG = +# a22_s4SQ 1 } in
       let { __DEFAULT ~ s#1_X39F
       <- writeFloatArray#
            arr#_i2Pd
            (+# a23_i30Y 1)
            (plusFloat#
               (plusFloat#
                  (plusFloat#
                     (plusFloat#
                        (plusFloat#
                           (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) 1)))
                           (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) (-1)))) __float -1.0))
                        (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG 1))) __float 2.0))
                     (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (-1)))) __float -2.0))
                  (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG (*# (-1) ipv1_X2LM)) 1))))
               (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
            s#_X39w
       } in .....
```

```
      0000163f movl  0x03(%edi),%ecx
      00001642 movl  0x07(%edi),%edx
      00001645 movl  0x08(%ebp),%esi
      00001648 movl  0x10(%ebp),%ebx
      0000164b movl  %ebx,0x04(%esp)
      0000164f leal  0x02(%esi,%edx),%eax
      00001653 movl  %eax,(%esp)
      00001656 movl  0x14(%ebp),%eax
      00001659 leal  0x02(%esi,%eax),%edi
      0000165d leal  (%esi,%eax),%ebx
      00001660 addl  %edx,%ebx
      00001662 addl  %edx,%edi
LOAD  00001664 movss 0x08(%ecx,%edi,4),%xmm1
LOAD  0000166a subss 0x08(%ecx,%ebx,4),%xmm1
      00001670 movl  (%esp),%edi
LOAD  00001673 movss 0x08(%ecx,%edi,4),%xmm2
      00001679 addss %xmm2,%xmm2
      0000167d addss %xmm1,%xmm2
      00001681 leal  (%edx,%esi),%edi
LOAD  00001684 movss 0x08(%ecx,%edi,4),%xmm1
      0000168a mulss %xmm0,%xmm1
      0000168e addss %xmm2,%xmm1
      00001692 leal  0x02(%esi),%edi
      00001695 movl  %edi,(%esp)
      00001698 movl  %edi,%ebx
      0000169a subl  %eax,%ebx
      0000169c addl  %edx,%ebx
LOAD  0000169e addss 0x08(%ecx,%ebx,4),%xmm1
      000016a4 movl  $0x3ffffff,%ebx
      000016a9 subl  %eax,%ebx
      000016ab leal  0x01(%ebx,%esi),%eax
      000016af addl  %edx,%eax
LOAD  000016b1 subss 0x08(%ecx,%eax,4),%xmm1
      000016b7 movl  0x04(%esp),%eax
      000016bb movl  0x14(%esp),%ecx
STORE 000016bf movss %xmm1,0x0c(%eax,%ecx,4)
```

```
      000016c5 movl  0x10(%ebp),%eax
      000016c8 movl  %eax,0x04(%esp)
      000016cc movl  0x14(%ebp),%edx
      000016cf leal  0x01(%esi,%edx),%ebx
      000016d3 movl  0x10(%esp),%edi
      000016d7 movl  0x03(%edi),%eax
      000016da movl  0x07(%edi),%ecx
      000016dd addl  %ecx,%ebx
      000016df leal  0x03(%esi,%edx),%edi
      000016e3 addl  %ecx,%edi
LOAD  000016e5 movss 0x08(%eax,%edi,4),%xmm1
LOAD  000016eb subss 0x08(%eax,%ebx,4),%xmm1
      000016f1 leal  0x03(%esi,%ecx),%edi
LOAD  000016f5 movss 0x08(%eax,%edi,4),%xmm2
      000016fb addss %xmm2,%xmm2
      000016ff addss %xmm1,%xmm2
      00001703 leal  0x01(%esi,%ecx),%edi
LOAD  00001707 movss 0x08(%eax,%edi,4),%xmm1
      0000170d mulss %xmm0,%xmm1
      00001711 addss %xmm2,%xmm1
      00001715 leal  0x03(%esi),%edi
      00001718 subl  %edx,%edi
      0000171a addl  %ecx,%edi
LOAD  0000171c addss 0x08(%eax,%edi,4),%xmm1
      00001722 leal  0x01(%esi),%edi
      00001725 subl  %edx,%edi
      00001727 addl  %ecx,%edi
LOAD  00001729 subss 0x08(%eax,%edi,4),%xmm1
      0000172f movl  0x04(%esp),%eax
      00001733 movl  0x14(%esp),%ecx
STORE 00001737 movss %xmm1,0x10(%eax,%ecx,4)
```

```
$wa4_s3HS =
  \ (ww4_s3lq :: Int#) (w2_s3ls :: State# RealWorld) ->
    case >=# (+# ww4_s3lq 4) ipv8_i30r of _ {
      False ->
        let { a22_s4SQ = +# ww4_s3lq (*# ww3_s3ly ipv1_X2LM) } in
        let { Vector rb_i2YQ _ rb2_i2YS ~ _ <- ds6_d2b5 `cast` ... } in
        let { a23_i30Y = +# ww4_s3lq (*# ww3_s3ly ipv1_X2LM) } in
       let { __DEFAULT ~ s#_X39w
       <- writeFloatArray#
            arr#_i2Pd
            a23_i30Y
            (plusFloat#
               (plusFloat#
                  (plusFloat#
                     (plusFloat#
                        (plusFloat#
                           (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) 1)))
                           (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) (-1)))) __float -1.0))
                        (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ 1))) __float 2.0))
                     (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ (-1)))) __float -2.0))
                  (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ (*# (-1) ipv1_X2LM)) 1))))
               (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
            (w2_s3ls `cast` ...)
       } in
       let { a24_s4TG = +# a22_s4SQ 1 } in
       let { __DEFAULT ~ s#1_X39F
       <- writeFloatArray#
            arr#_i2Pd
            (+# a23_i30Y 1)
            (plusFloat#
               (plusFloat#
                  (plusFloat#
                     (plusFloat#
                        (plusFloat#
                           (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) 1)))
                           (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) (-1)))) __float -1.0))
                        (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG 1))) __float 2.0))
                     (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (-1)))) __float -2.0))
                  (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG (*# (-1) ipv1_X2LM)) 1))))
               (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
            s#_X39w
       } in .....
```

```
fillLine4 !x
 | x + 4 > x1      = fillLine1 x
 | otherwise
 = do let srcCur0 = make  (Z:.y:.x)
      let srcCur1 = shift (Z:.0:.1) srcCur0
      let srcCur2 = shift (Z:.0:.1) srcCur1
      let srcCur3 = shift (Z:.0:.1) srcCur2

      let val0    = load srcCur0
      let val1    = load srcCur1
      let val2    = load srcCur2
      let val3    = load srcCur3

      let !dstCur0 = x + y * width
      unsafeWrite vec (dstCur0)     val0
      unsafeWrite vec (dstCur0 + 1) val1
      unsafeWrite vec (dstCur0 + 2) val2
      unsafeWrite vec (dstCur0 + 3) val3
      fillLine4 (x + 4)
```

The poison

```
touch# :: forall o
        .  o -> State# RealWorld
             -> State# RealWorld
```

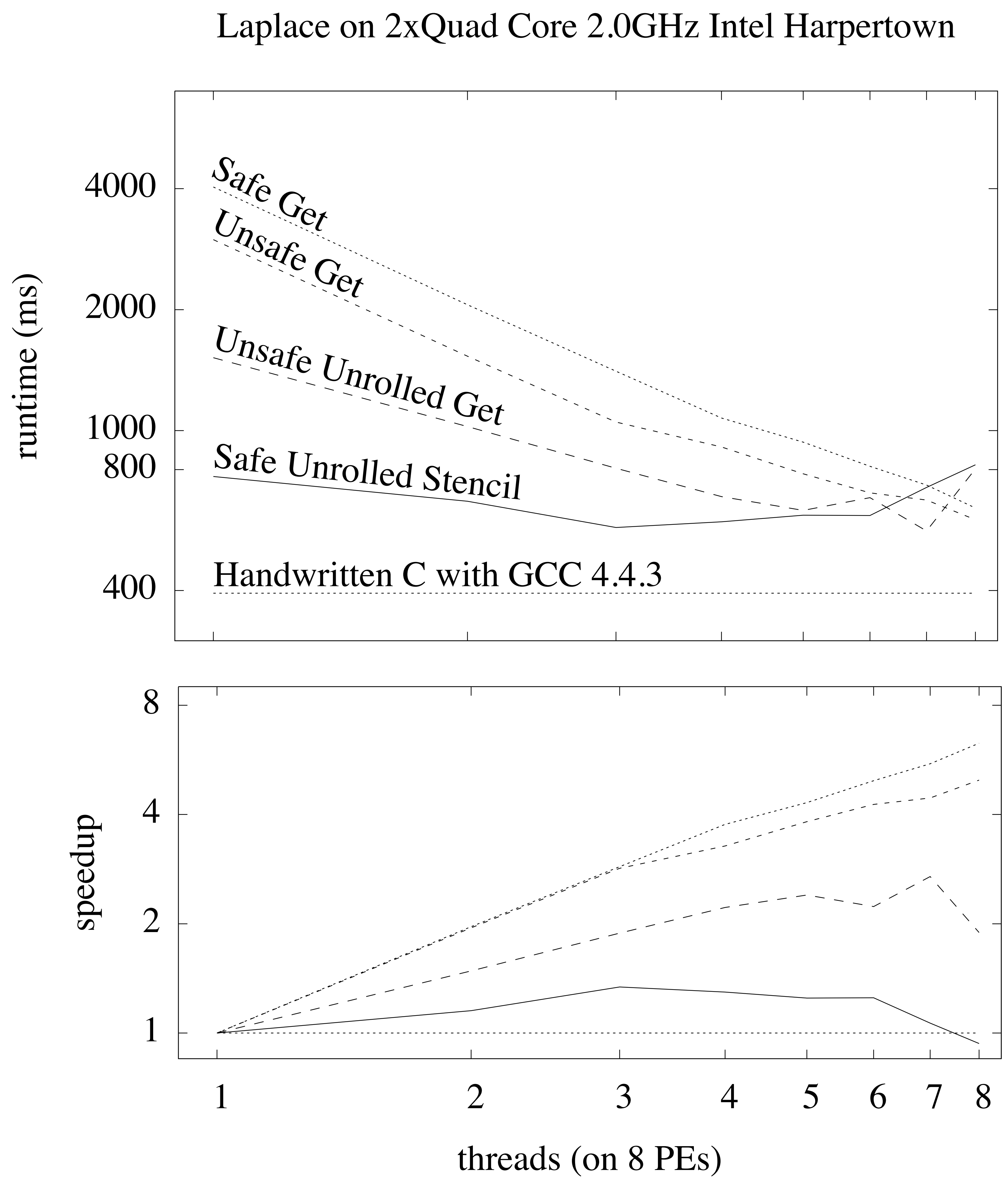
- Quantifier **`forall o.`** is “special”..
- You can instantiate it to unboxed types.

```
fillLine4 !x
 | x + 4 > x1      = fillLine1 x
 | otherwise
 = do let srcCur0 = make  (Z:.y:.x)
      let srcCur1 = shift (Z:.0:.1) srcCur0
      let srcCur2 = shift (Z:.0:.1) srcCur1
      let srcCur3 = shift (Z:.0:.1) srcCur2

      let val0    = load srcCur0
      let val1    = load srcCur1
      let val2    = load srcCur2
      let val3    = load srcCur3

      touch val0 ; touch val1 ; touch val2 ; touch val3

      let !dstCur0 = x + y * width
      unsafeWrite vec (dstCur0)     val0
      unsafeWrite vec (dstCur0 + 1) val1
      unsafeWrite vec (dstCur0 + 2) val2
      unsafeWrite vec (dstCur0 + 3) val3
      fillLine4 (x + 4)
```

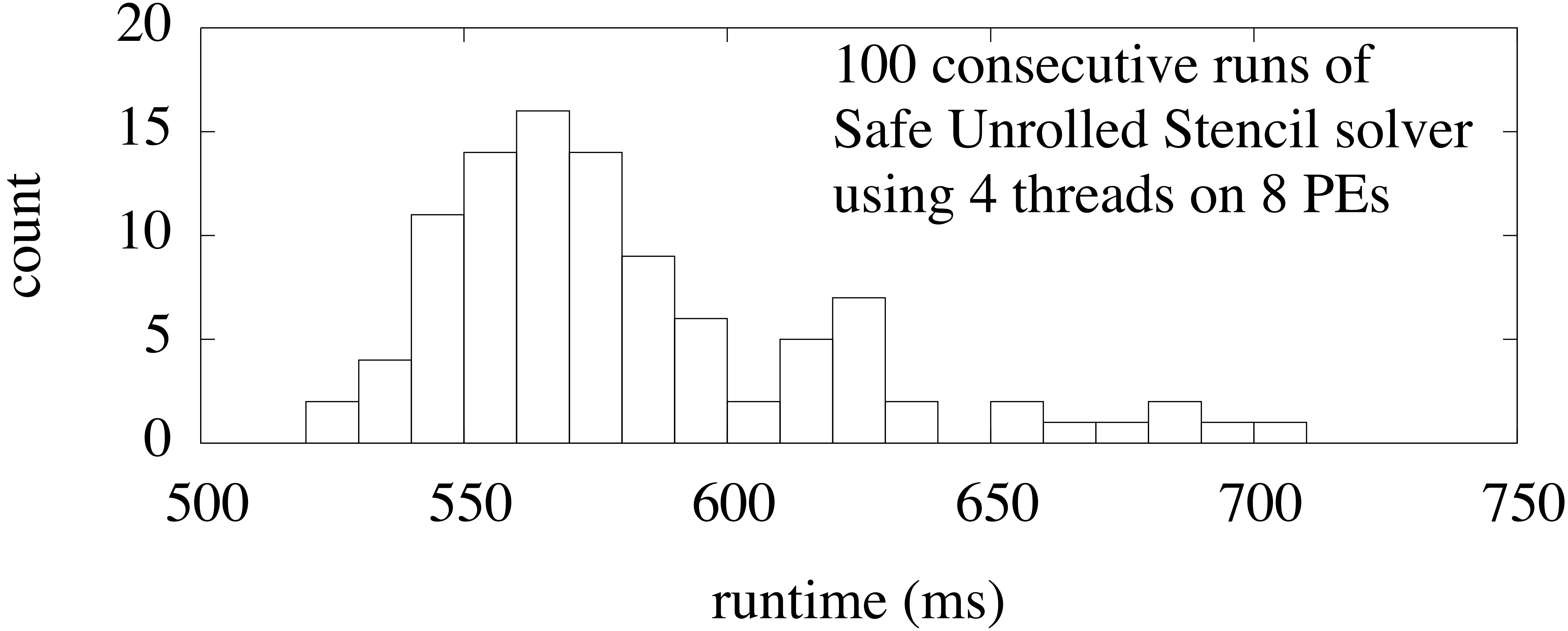
```
  9b0: mov     0x2e(rbx), rcx
  9b4: mov     0x1e(rbx), rdx
  9b8: mov     rdx, rsi
  9bb: imul    rcx, rsi
  9bf: mov     0x36(rbx), rdi
  9c3: lea     0x4(r14,rdi,1), r8
  9c8: add     r14, rdi
  9cb: lea     0x1(rcx), r9
  9cf: imul    rdx, r9
  9d3: lea     0x2(r9,rdi,1), r10
  9d8: mov     0x6(rbx), r11
  9dc: mov     0xe(rbx), r15

● 9e0: movss   0x10(r15,r10,4), xmm7
  9e7: lea     (r8,r9,1), r10
● 9eb: movss   0x10(r15,r10,4), xmm8
  9f2: subss   xmm7, xmm8
  9f7: lea     (r8,rsi,1), r10
● 9fb: movss   0x10(r15,r10,4), xmm9
  a02: addss   xmm9, xmm9
  a07: addss   xmm8, xmm9
  a0c: lea     0x2(rsi,rdi,1), r10
● a11: movss   0x10(r15,r10,4), xmm8
  a18: movaps  xmm8, xmm10
  a1c: mulss   xmm0, xmm10
  a21: addss   xmm9, xmm10
  a26: dec     rcx
  a29: imul    rdx,rcx
  a2d: add     rcx,r8
```

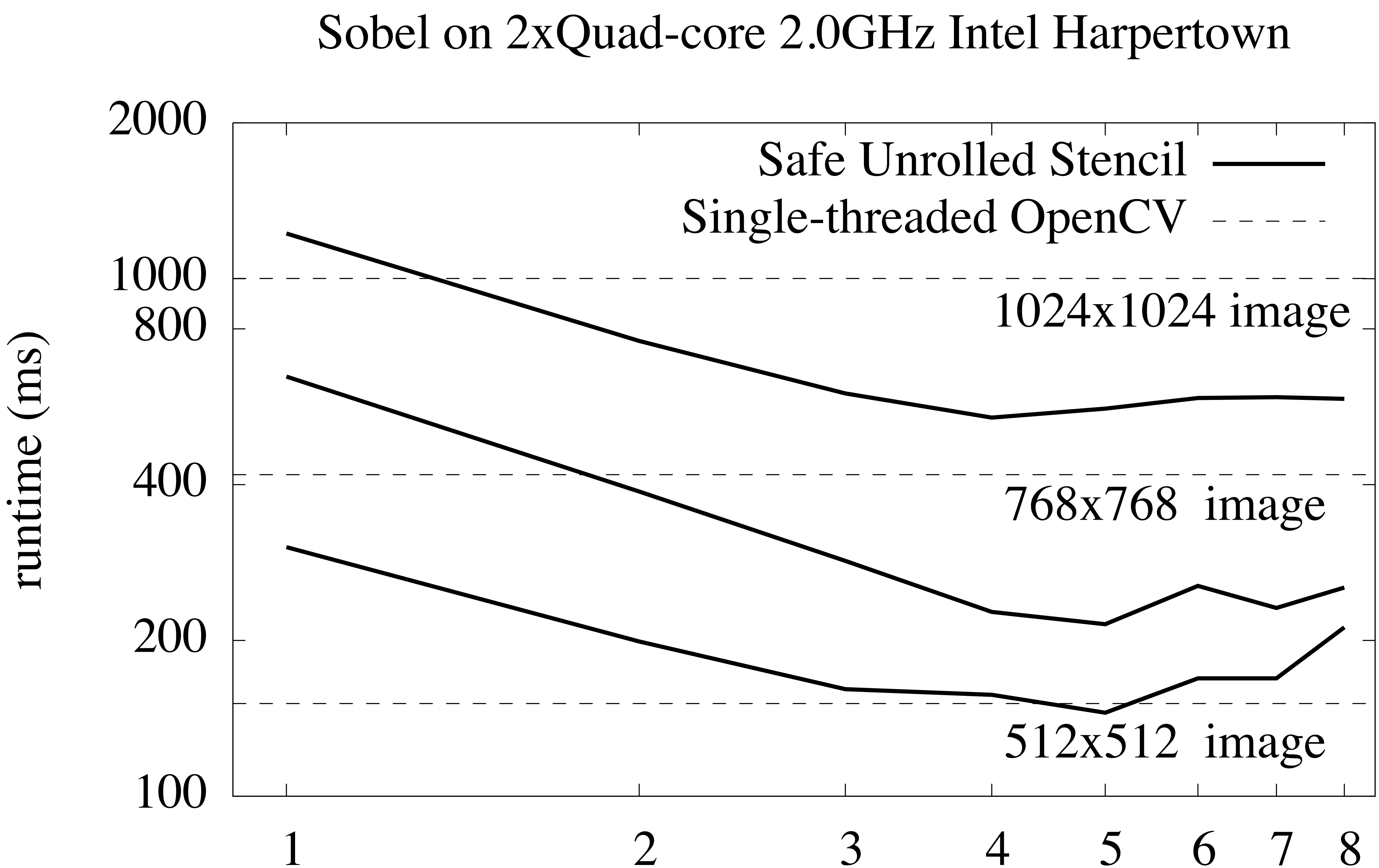
```
● a30: addss   0x10(r15,r8,4), xmm10
  a37: lea     0x1(r9,rdi,1), rdx
● a3c: movss   0x10(r15,rdx,4), xmm9
  a43: lea     0x3(r9,rdi,1), rdx
● a48: movss   0x10(r15,rdx,4), xmm11
  a4f: subss   xmm9, xmm11
  a54: lea     0x3(rsi,rdi,1), rdx
● a59: movss   0x10(r15,rdx,4), xmm12
  a60: addss   xmm12, xmm12
  a65: addss   xmm11, xmm12
  a6a: lea     0x1(rsi,rdi,1), rdx
● a6f: movss   0x10(r15,rdx,4), xmm11
  a76: movaps  xmm11, xmm13
  a7a: mulss   xmm0, xmm13
  a7f: addss   xmm12, xmm13
  a84: lea     0x3(rcx,rdi,1), rdx
● a89: addss   0x10(r15,rdx,4), xmm13

  a90: lea     (rdi,r9,1), rdx
● a94: subss   0x10(r15,rdx,4), xmm7
  a9b: addss   xmm8, xmm8
  aa0: addss   xmm7, xmm8
  aa5: lea     0x1(rcx,rdi,1), rdx
  aaa: lea     0x2(rcx,rdi,1), r8
  aaf: lea     (rdi,rsi,1), r10
● ab3: movss   0x10(r15,r10,4), xmm7
  aba: mulss   xmm0, xmm7
  abe: addss   xmm8, xmm7
● ac3: movss   0x10(r15,r8,4), xmm8
  aca: addss   xmm8, xmm7
  acf: lea     (rdi,rcx,1), r8
● ad3: subss   0x10(r15,r8,4), xmm7
```

```
  ada: add     rax, rdi
  add: add     rdi, r9
● ae0: subss   0x10(r15,r9,4), xmm9
  ae7: addss   xmm11, xmm11
  aec: addss   xmm9, xmm11
  af1: lea     (rdi,rsi,1), r8
● af5: movss   0x10(r15,r8,4), xmm9
  afc: mulss   xmm0, xmm9
  b01: addss   xmm11, xmm9
● b06: movss   0x10(r15,rdx,4), xmm11
  b0d: addss   xmm11, xmm9
  b12: add     rcx, rdi
● b15: subss   0x10(r15,rdi,4), xmm9

  b1c: add     r14,rsi
◇ b1f: movss   xmm9,0x10(r11,rsi,4)
  b26: mov     0x6(rbx),rcx
◇ b2a: movss   xmm7,0x14(rcx,rsi,4)
  b30: subss   xmm11,xmm13
  b35: mov     0x6(rbx),rcx
◇ b39: movss   xmm13,0x18(rcx,rsi,4)
  b40: subss   xmm8,xmm10
  b45: mov     0x6(rbx),rcx
◇ b49: movss   xmm10,0x1c(rcx,rsi,4)
  b50: lea     0x8(r14),rcx
  b54: lea     0x4(r14),r14
  b58: cmp     0x26(rbx),rcx
  b5c: jle     9b0
```

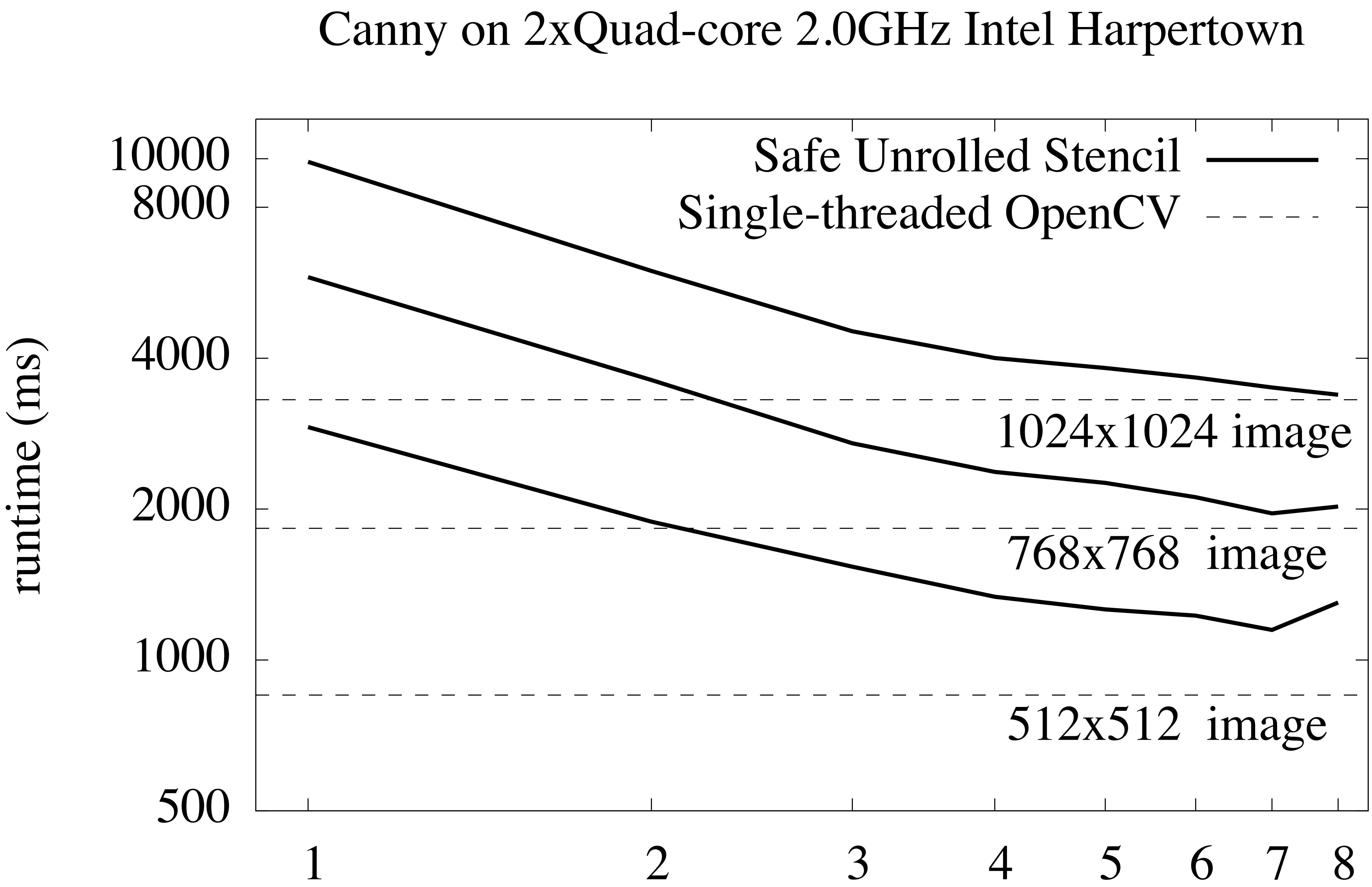


Laplace on 2xQuad Core 2.0GHz Intel Harpertown
runtime (ms)
4000
2000
1000
800
400
Safe Get
Unsafe Get
Unsafe Unrolled Get
Safe Unrolled Stencil
Handwritten C with GCC 4.4.3
speedup
8
4
2
1
1 2 3 4 5 6 7 8
threads (on 8 PEs)

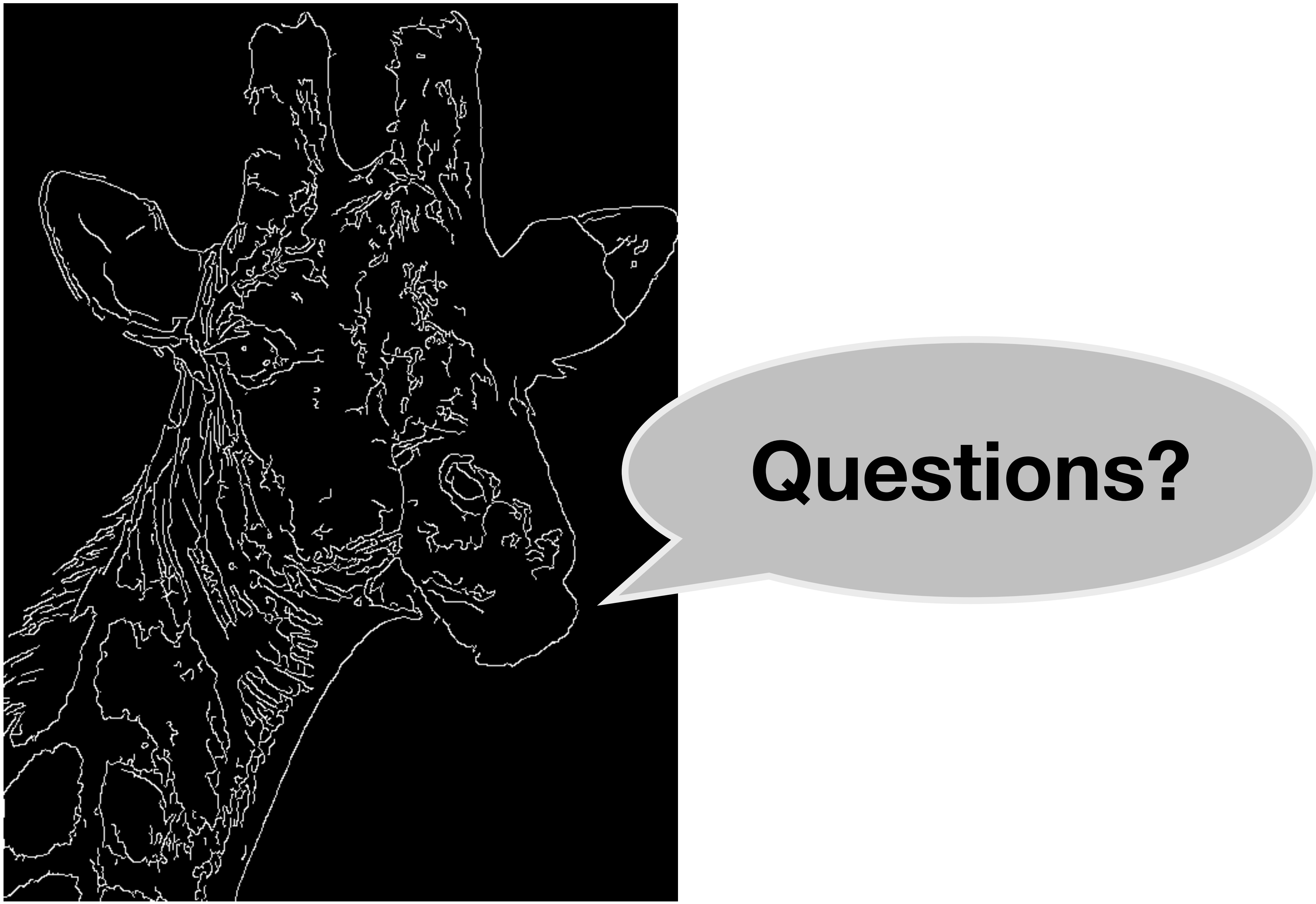

100 consecutive runs of
Safe Unrolled Stencil solver
using 4 threads on 8 PEs
count
runtime (ms)
0
5
10
15
20
500
550
600
650
700
750



Sobel on 2xQuad-core 2.0GHz Intel Harpertown
Safe Unrolled Stencil
Single-threaded OpenCV
1024x1024 image
768x768 image
512x512 image
runtime (ms)
2000
1000
800
400
200
100
1
2
3
4
5
6
7
8


Canny on 2xQuad-core 2.0GHz Intel Harpertown
Safe Unrolled Stencil
Single-threaded OpenCV
1024x1024 image
768x768 image
512x512 image
runtime (ms)
10000
8000
4000
2000
1000
500
1
2
3
4
5
6
7
8

	GCC 4.4.3 OpenCV	GHC 7.0.2 + Repa with # threads 1	2	4	8
Grey scale	10.59	12.05	6.19	3.25	2.08
Gaussian blur	3.53	17.42	9.70	5.92	5.15
Detect	18.95	68.73	43.81	31.21	28.49
Differentiate	fused	11.90	7.41	5.38	5.22
Mag / Orient	fused	27.09	16.11	10.45	7.85
Maxima	fused	12.87	7.84	4.83	3.32
Select strong	fused	10.01	5.68	3.60	5.16
Link edges	fused	6.86	6.77	6.95	6.94
TOTAL	33.05	98.25	59.70	40.38	35.72


Questions?