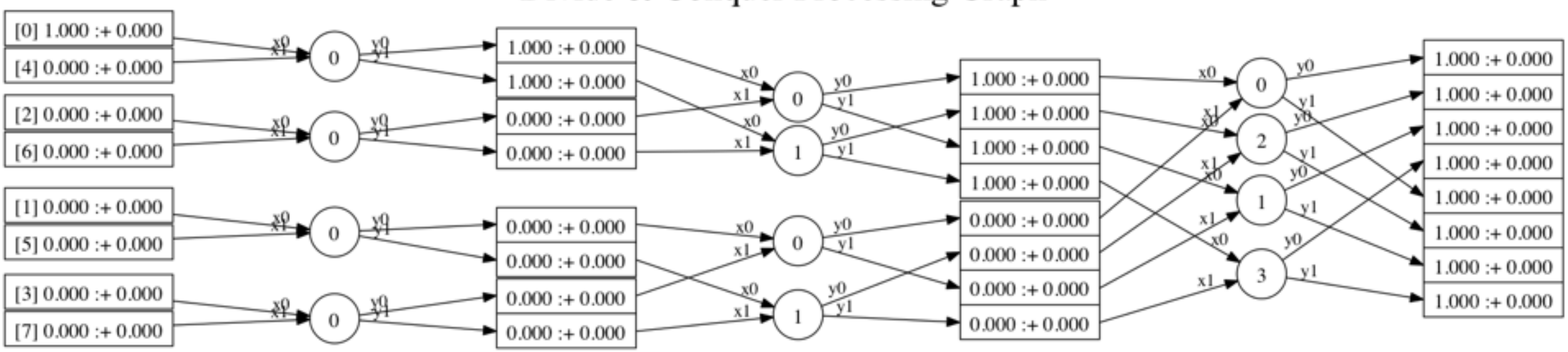
FFT via Circat

Dave <capn.freako@gmail.com>
&
Conal <conal@conal.net>

Haskell Meetup
Hacker Dojo
Mountain View, CA
October 15, 2015

A year and a half ago…

Divide & Conquer Processing Graph



0: y0 = (1.000 :+ 0.000) * x0 + (1.000 :+ 0.000) * x1 y1 = (1.000 :+ 0.000) * x0 + (-1.000 :+ 0.000) * x1

1: y0 = (1.000 :+ 0.000) * x0 + (0.000 :+ -1.000) * x1 y1 = (1.000 :+ 0.000) * x0 + (0.000 :+ 1.000) * x1

2: y0 = (1.000 :+ 0.000) * x0 + (0.707 :+ -0.707) * x1 y1 = (1.000 :+ 0.000) * x0 + (-0.707 :+ 0.707) * x1

3: y0 = (1.000 :+ 0.000) * x0 + (-0.707 :+ -0.707) * x1 y1 = (1.000 :+ 0.000) * x0 + (0.707 :+ 0.707) * x1

Computational Node Legend

FFT in Haskell

```
radix2_DIT :: RealFloat a =>
  [Complex a] -> [Complex a]
radix2_DIT []     = []
radix2_DIT [x]    = [x]
radix2_DIT xs     = (++) (zipWith (+) xes xos)
                         (zipWith (-) xes xos)
    where xes = radix2_DIT (evens xs)
          xos = zipWith (*)
                   (radix2_DIT (odds xs))
                   [ wn ** (fromIntegral k)
                       | k <- [0..]]
          wn  = exp ( 0.0 :+ ( -2.0 * pi / n))
          n   = fromIntegral (length xs)
```

A useful pursuit, but…

- Assumes container will always be a list.
- What about trees? Or, other traversable structures?
- How about defining FFT for 3 primitives?:
  - Id
  - Pair
  - (:.) (functor composition)

Enter *Circat*…

- Conal Elliott's machinery for representing circuits (and other things), using *Cartesian Closed Categories*.

- Contains some very useful data structures, and higher order functions on them, for doing FFT (and other things).

- We'll use: *RTree*, which is a perfect binary leaf tree parameterized by depth, because it naturally enforces a balanced, log-2 breakdown of computation and provides some very elegant decimation mechanisms.

First, the answer…

```haskell
-- Phasor, as a function of tree depth.
phasor :: (IsNat n, RealFloat a, Enum a) =>
    Nat n -> RTree n (Complex a)
phasor n = scanlTEx (*) 1 (pure phaseDelta)
    where phaseDelta = cis ((-pi) / 2 ** natToZ n)

-- Radix-2, DIT FFT
fft_r2_dit :: (IsNat n, RealFloat a, Enum a) =>
    RTree n (Complex a) -> RTree n (Complex a)
fft_r2_dit = fft_r2_dit' nat

fft_r2_dit' :: (RealFloat a, Enum a) =>
    Nat n -> RTree n (Complex a) -> RTree n (Complex a)
fft_r2_dit'  Zero     = id
fft_r2_dit' (Succ n) = toB
                     . inP (uncurry (+) &&& uncurry (-))
                     . secondP (liftA2 (*) (phasor n))
                     . fmap (fft_r2_dit' n)
                     . bottomSplit
```

Second, some comparisons…

- Old:

```
[ wn ** (fromIntegral k) | k <- [0..]]
 where wn  = exp ( 0.0 :+ ( -2.0 * pi / n))
       n   = fromIntegral (length xs)
```

- New:

```
-- Phasor, as a function of tree depth.
phasor :: (IsNat n, RealFloat a, Enum a) =>
    Nat n -> RTree n (Complex a)
phasor n = scanlTEx (*) 1 (pure phaseDelta)
 where phaseDelta = cis ((-pi) / 2 ** natToZ n)
```

FFT as a class:* phasor*, like* pure*, could be overloaded.

- Old:

```
radix2_DIT []     = []
radix2_DIT [x]    = [x]
radix2_DIT xs     = (++) (zipWith (+) xes xos)
                         (zipWith (-) xes xos)
    where xes = radix2_DIT (evens xs)
          xos = zipWith (*)
                  (radix2_DIT (odds xs))
                  {phasor}
```

- New:

```
-- Radix-2, DIT FFT
fft_r2_dit'  Zero     = id
fft_r2_dit' (Succ n) =
    toB
    . inP (uncurry (+) &&& uncurry (-))
    . secondP (liftA2 (*) (phasor n))
    . fmap (fft_r2_dit' n)
    . bottomSplit
```

Description has been elevated in its level of abstraction.

Future Directions

- FFT as a class
- Defined instances for:
  - Id
  - Pair
  - (:.) (functor composition)
- FFT of higher order structures *derived* from above.

Questions?

Thank you!

References

TreeViz: https://wiki.haskell.org/Treeviz

Circat: https://github.com/conal/circat

Lambda-CCC: https://github.com/conal/lambda-ccc

FFT: https://en.wikipedia.org/wiki/Fast_Fourier_transform