Efficient Numerical Code in Haskell

Jyotirmoy Bhattacharya

Ambedkar University Delhi

Delhi NCR Haskell Meetup, June 2016

About

- Email: `jyotirmoy@jyotirmoy.net`
- Website: `https://www.jyotirmoy.net`
- GitHub: `https://github.com/jmoy`

Lists: The Good

The List datatype

```
data List a = Empty
            | Node a (List a)
```

- Lazy links can represent infinite and self-referential structures:

```
fib = 1:1:zipWith (+) fib (tail fib)
```

- Lazy data can support modularity:

```
filter (even . fst) $ map (\i->(i,i^2)) [1..]
```

- Updates can share representation with old values:

```
x = "cake"
y = 'b':tail x
```

Lists: The Bad

- Access to the n-th elements in $O(n)$.
- High memory overhead and poor data locality.
- Pointer chasing for traversal.
- Data and links must be forced before use.

Sieve of Eratosthenes (Python)

```python
import numpy as np
def count_primes(n):
  "Count no. of primes smaller than n"
  v = np.full(n,True,np.bool)
  # We want to set v[i] = False for non-primes
  v[0] = v[1] = False

  for p in range(n):
    if v[p]:
      # Found a prime
      # Mark all its multiples as non-prime
      for k in range(p*p,n,p):
        v[k] = False
  return np.count_nonzero(v)
```

The `vector` package

- Module `Data.Vector`. Array of heap-allocated values.
- Module `Data.Vector.Unboxed`. Array of unboxed values.
- Module `Data.Vector.Storable`. Array allocated in memory that can be passed to foreign functions.

Sieve of Eratosthenes (Haskell)

```haskell
import qualified Data.Vector.Unboxed as V
import Data.Vector.Unboxed ((//),(!))
countPrimes n = V.length $ V.filter id $ loop 2 mask0
  where
    mask0 = (V.replicate n True) // [(0,False),(1,False)]
    loop p mask
      | p == n = mask
      | not (mask ! p) = loop (p+1) mask
      | otherwise = loop (p+1) mask'
        where
          mask' = mask // [(k,False)|k<-[p*p,p*p+p..(n-1)]]
```

The `ST` Monad

- A value of type `(ST s a)` is a computation which transforms a state of type `s` and additionally produces a value of type `a`.
- The state is not accessible to library users. So they cannot duplicate it.
- Action `newSTRef` creates a new *mutable variable* within a state to which values can be written through `writeSTRef` and `modifySTRef` and read through `readSTRef`.
- The `runST` function takes a computation in `ST` and gives us the value produced. This is what allows us to embed `ST` computation in pure programs.

Stupid ST example

```haskell
import Control.Monad.ST
import Data.STRef
import Control.Monad (forM_)

lengthJB::[a]->Int
lengthJB xs = runST $ do
  ctr <- newSTRef 0
  forM_ xs $ \_ -> modifySTRef ctr (+ 1)
  readSTRef ctr
```

Simulating a bank

The External Interface

```
type Name = String
type Ledger = [(Name,Double)]
data Transaction = Transaction
                     Name     -- From
                     Name     -- To
                     Double   -- Amount

-- How do we implement this efficiently?
simulateBank::Ledger->[Transaction]->Ledger
```

Simulating a bank (contd.)

```haskell
-- The state of a bank
type Bank s = HashMap Name (STRef s Double)
-- Create a bank reflecting a given ledger
mkBank :: [(Name,Double)] -> ST s (Bank s)
-- Modify the state of a bank to reflect a transaction
transact :: Bank s -> Transaction -> ST s ()
-- Turn the state of the bank into a ledger
readLedger :: Bank s -> ST s Ledger

simulateBank::Ledger -> [Transaction] -> Ledger
simulateBank ledger ts = runST $ do
  b <- mkBank ledger
  mapM_ (transact b) ts
  readLedger b
```

Simulating a bank (contd.)

```
mkBank ledger = do
  let (n,as) = unzip ledger
  vs <- mapM newSTRef as
  return $ fromList $ zip n vs

transact b (Transaction cFrom cTo amt) = do
      modifySTRef' (b ! cFrom) (subtract amt)
      modifySTRef' (b ! cTo) (+ amt)

readLedger b = do
  let (n,v) = unzip $ toList b
  as <- mapM readSTRef v
  return $ zip n as
```

The ST Monad (continued)

- The type of `runST` is

  ```
  runST :: (forall s. ST s a) -> a
  ```

  This is so that references to mutable variables cannot escape a `runST` and the computation encapsulated by it really is pure.
- Neither

  ```
  runST $ newSTRef 0
  ```

  nor

  ```
  \p -> runST $ readSTRef p
  ```

  are accepted by the typechecker.

Mutable Vectors

- The `vector` package provides mutable vectors (`MVector`) which can be created, accessed and destructively modified in the `ST` monad (also variants for the `IO` monad).
- Like immutable vectors there are unboxed and storable variants.
- Conversion between mutable and immutable vectors is possible, but may copy the vector

```
freeze :: MVector s a -> ST s (Vector a)
thaw :: Vector a -> ST s (MVector s a)
```

Sieve of Eratosthenes (again)

```haskell
import qualified Data.Vector.Unboxed as VI
import qualified Data.Vector.Unboxed.Mutable as V
countPrimes::Int->Int
countPrimes n
  = VI.length $ VI.filter id $ mask
  where
    mask = VI.create mkMask
    mkMask::ST s (V.MVector s Bool)
    mkMask = do
      v <- V.replicate n True
      V.write v 0 False
      V.write v 1 False
      forM_ [2..(n-1)] $ \p -> do
        b <- V.read v p
        when b $
          forM_ [p*p,p*p+p..(n-1)] $ \i ->
            V.write v i False
      return v
```

REPA: REgular PArallel arrays

- Key data type

  ```
  Array r sh e
  ```

- `r` can be `D`, `U` `F` and others.
- `sh` is a type of the shape class `Shape`. Example types

  ```
  Z :. Int :. Int
  ```

  Example values

  ```
  Z :. 0 :. 1
  ```

- Moving from a delayed to manifest represenation is under the programmer's control: `computeP`, `computeS`.

Parallelization for free

```
ubuntu@ip-172-31-40-69:~/examples-repa /usr/bin/time \
./dist/build/examples-repa/examples-repa \
    80 20 Wave.jpg WaveEC2.jpg +RTS -N1
9.59user 0.11system 0:09.70elapsed 100%CPU
(0avgtext+0avgdata 556080maxresident)k
0inputs+43824outputs (0major+2447minor)pagefaults 0swaps

ubuntu@ip-172-31-40-69:~/examples-repa /usr/bin/time \
    ./dist/build/examples-repa/examples-repa \
    80 20 Wave.jpg WaveEC2.jpg +RTS -N4
14.09user 0.14system 0:04.66elapsed 305%CPU
(0avgtext+0avgdata 559488maxresident)k
0inputs+43824outputs (0major+2910minor)pagefaults 0swaps
```