Assignment 1
Programming Massively Parallel Hardware

Mads Thoudahl /qmh332

September 11, 2015

1 Task 1 - Proof

Proving the equality of the equation:

$$
\begin{aligned}
& redomap(\odot) f e_{\odot} && (1)\\
=& (redomap(\odot) e_{\odot}).(mapf) && (2)\\
=& (redomap(\odot) e_{\odot}).(mapf).id && (3)\\
=& (redomap(\odot) e_{\odot}).(mapf).(reduce(++)[\,]).distr_p && (4)\\
\underset{LHI2}{=}& (reduce(\odot) e_{\odot}).(reduce(++)[\,]).(map(mapf)).distr_p && (5)\\
\underset{LHI3}{=}& (reduce(\odot) e_{\odot}).((map) reduce(\odot) e_{\odot})).distr_p && (6)
\end{aligned}
$$

2 Task 2 - Longest Satisfying Segment Problem

The assignment had to be extended to work as supposed, as `firstx` always took the value 0 thereby making the neutral element nonneutral... The newvalues are all dependent on the fact that the two sublists are connected.

```
lssop (lssx, lisx, lcsx, tlx, firstx, lastx, okx)
(lssy, lisy, lcsy, tly, firsty, lasty, oky) =
  (newlss, newlis, newlcs, tlx+tly, fx, ly, newok)
    connect = if (tlx==0 || tly==0) then True else p [lastx,firsty]
    newlss  = lssx `max` lssy `max` (if connect then (lcsx + lisy) else 0)
    newlis  = if (okx && connect) then (tlx + lisy) else lisx
    newlcs  = if (oky && connect) then (tly + lcsx) else lcsy
    newok   = okx && oky && connect
    fx      = if (tlx==0) then firsty else firstx
    ly      = if (tly==0) then lastx else lasty
```

This code fulfills its purposes in my simple test-cases.

3 Task 3

This task has proven to be very hard indeed. I have been unsuccesful in solving the task, but i have included some code to try and illustrate my thoughts on the problem.

My idea was to zip up the data values along with the indices they should be moved to, thus calculating their offset (did that in `idxoffsets`) and their local index... the local idx are in my head some iota from [0..predicate trues], and from [predicate falses-n..n], but i cannot figure out how to calculate those local indices.

```
segmSpecialFilter :: (a->Bool) -> [Int] -> [a] -> ([Int],[a])
segmSpecialFilter cond sizes arr =
    --sizes [3,0,0,4,0,0,0]  arr [3,2,1,2,3,4,5]]
    let n          = length arr
        isflag     = map (\x -> if x==0 then 0::Int else 1::Int) sizes
                                          -- [1,0,0,1,0,0,0]
        condtest   = map cond arr         -- [T,F,T,F,T,F,T]
        istrue     = map (\b -> if b==True then 1::Int else 0::Int) condtest
                                          -- [1,0,1,0,1,0,1]
        isfalse    = map (\b -> if b==True then 0::Int else 1::Int) condtest
                                          -- [0,1,0,1,0,1,0]
        segtrueacc = segmScanInc (+) 0 sizes istrue
                                          -- [1,1,2,0,1,1,2]
        -- segTrues   = segmReduce (max) 0 sizes segtrueacc -- does not compile
                                          -- [2,2]

        idxs       = scanExc (+) 0 sizes     -- [0,3,3,3,7,7,7]
        idxoffset  = zipWith (*) isflag idxs -- [0,0,0,3,0,0,0]
        idxoffsets = scanInc (+) 0 idxoffset -- [0,0,0,3,3,3,3]
        prevlocidx = zipWith (-) (reduce (++) [] (map (\f -> [1..f]) flag)) (replicate n 1)
                                          -- [0,1,2,0,1,2,3]
        localidx   = replicate n 0  -- bogus!
        -- goal : flag [3,0,0,4,0,0,0] countflags [2,0,1,2,0,2,0] filteredelms [3,1,2,3,5,2,4]
        newidx     = zipWith (+) idxoffsets localidx
        newelm     = permute newidx arr
    in  (isflag, newelm)
```

further, I have attached the relevant haskell file in the zip file along with the handin.

4 Task 4 - CUDA programming

The code is implemented in the attached zip file under the `cuda` directory as `ass1.cu` and with a makefile as described. It has some global variables like number of threads and whether to time memory transfers set in define clauses in the file.

4.1 timings

Table 1: timings - avg of 3 runs all on gpu01 at diku server

800.000 calcs	GPU	CPU
with MEM	3.11 ms	3.19 ms
without MEM	0.165 ms	3.26 ms

All calculations are within a fault tolerance (EPSILON) of 0.0005. It seems that with the current calculations, around 14200 calculations make it break even. But if you do count the memory transfers in, it barely break even at 800.000 calculations. There is a define statement in the code to switch timing of memory transfers.