# Asignment 1: Tycho Canter Cremers

# 1.

(reduce (+) ne) . map f . reduce ++ [] . distrP

rule 2

(reduce (+) ne) . reduce(++) [] . map(map f() ) . distrP

rule 3

(reduce (+) ne) . (map(reduce (+) ne)) . map(map f()) . distrP

rule 1: map f map g

(reduce (+) ne) . map (reduce (+) ne . map f() ) . distrP

res <- [.,.,.] <- [[f(...)][f(...)][f(...)]] <-[[...][...][...]]

# 2.

Below is the shell timed output for the lssp code run with an array of 8 million entries. First the code compiled with futhark-opencl and last the code compiled with futhark-c.

*[mns267@a00332 PMPH-weekly1]$ futhark-dataset --i32-bounds=-20:20 -g [8000000]i32 | ./lssp -r 10 -t /dev/stderr*

*1785*

*1717*

*1767*

*1720*

*1711*

*1707*

*1711*

*1721*

*1699*

*1928*

*11i32*

*[mns267@a00332 PMPH-weekly1]$ futhark-c lssp.fut*

*[mns267@a00332 PMPH-weekly1]$ futhark-dataset --i32-bounds=-20:20 -g [8000000]i32 | ./lssp -r 10 -t /dev/stderr*

*16719*

*16656*

*16718*

*16664*

*16711*

*16658*

*16753*

*16659*

*16715*

*16653*

*11i32*

The GPU compiled code shows almost a 10 times speed increase over the CPU compiled code.

# 3.

I unfortunately haven’t had the chance to implement the flattened code with futhark but I will provide the (hopefully correct) pseudo code. I will try to implement it when I’m finished with all other assignments.

**Nested parralel code:**

*sqrt\_primes = primesOpt (sqrt (fromIntegral n))*

*nested = map (np -> let m = (n `div` p)*

*in map (nj -> j\*p) [2..m]*

*) sqrt\_primes*

*not\_primes = reduce (++) [] nested*

**Flattened code:**

*ms = map(\ P -> n / P) sgrt.primes*

*let mm1s = map (\m -> m-1) ms in*

*iots = F (map(\mm1 -> iota mm1) mm1s)*

*inds = scan-exc (+) 0 mm1s*

*size = reduce (+) 0 mm1s*

*flag = scatter (replicate size, 0) inds mm1s*

*tmp = replicate size 1*

*iots = sgment-scan-exc (+) offs tmp*

*twoms = map(\x->x+z)iots*

*rps = F (map(\mm1 P -> replicate mm1 P) mm1s sqrt prime)*

*nested = F(map(\js Ps ->map(\*) js ps) twoms rps)*

*= map(\*) twoms rps*

*4.*

Running the code prints the array size, the time it took the GPU to execute the kernel and the time the CPU took to execute the sequential function. Additionally, it prints the first 10 element of GPU and CPU output. And finally it checks if both output arrays are equal and prints the answer to that question.

*[mns267@a00332 ~]$ make run*

*./task4*

*Array size:32757*

*GPU took 33 microseconds (0.03ms)*

*-0.000000 - -0.455166 - -296.296295 - 78.717201 - 13.026664 - 6.350658 - 4.264308 - 3.303699 - 2.764683 - 2.423835 -*

*CPU took 390 microseconds (0.39ms)*

*-0.000000 - -0.455166 - -296.296295 - 78.717201 - 13.026664 - 6.350658 - 4.264308 - 3.303699 - 2.764683 - 2.423835 -*

*VALID*