Weekly 4

Christian Påbøl

December 20, 2020

Contents

1 TODO Task 1

2 Task 2

In lecture 8, on slide 41 we are given the mathematical formulas for calculating the adjoint values of a `scan (*)` statement. I wrote this up in futhark. I also inlined the partial differentials. Setting the last in lines 3 and 4 is done using an if in a map. Had this been written with a less strict type system in mind, i might have either used list concatenation or a "shift" operation, however futhark doesn't like manual list operations *nor* operations on only one index. Therefore, it is a map

```
1  let scanAdjoint [n] (as: [n]real) (ys_bar_0: [n]real) : [n]real =
2    let ys = scan (*) 1.0 as
3    let is = iota n
4    let cs = map (\i -> if i == n-1 then 1 else as[i+1]) is
5    let ds = map (\i -> if i == n-1 then 0 else ys_bar_0[i]) is
6    let ys_bar = scan lin_o (0, 1) (zip (reverse ds) (reverse cs))
7      |> map (\ (d, c) -> d + (c * ys_bar_0[n-1]) )
8      |> reverse
9    in map (\i -> if i == 0 then ys_bar[0] else (ys[i-1]*ys_bar[i])) is
```

It does validate.

3 Task 3

We are given the following code

```
forall i = 0 .. N-1 do
  forall j = 0 .. N-1 do
    let ps = map (*) A[i,:] B[:,j]
    let c = reduce (+) 0.0 ps

    let r_bar = C[i,j]
    let ps_bar = replicate N 0.0 -- init

    let ps_bar = map (+r) ps_bar

    forall k = 0 .. N-1 do
      let A_bar[i,k] += B[k,j] * ps_bar[k]
      let B_bar[k,j] += A[i,k] * ps_bar[k]
```

I look at this and see that i am supposed to rewrite this as two nested maps, which the compiler can optimize with tiling. I am also given a rough map of what transformations to apply. Starting with DBE:

- `c` is never used. Yeet that shizzle
- `ps` is redundant also, get outta here

```
forall i = 0 .. N-1 do
  forall j = 0 .. N-1 do

    let r_bar = C[i,j]
    let ps_bar = replicate N 0.0

    let ps_bar = map (+r_bar) ps_bar

    forall k = 0 .. N-1 do
      let A_bar[i,k] += B[k,j] * ps_bar[k]
      let B_bar[k,j] += A[i,k] * ps_bar[k]
```

We then simplify the definition of `ps_bar` and inline `r`

```
forall i = 0 .. N-1 do
  forall j = 0 .. N-1 do
```

```
    let ps_bar = replicate N C[i,j]
    forall k = 0 .. N-1 do
      let A_bar[i,k] += B[k,j] * ps_bar[k]
      let B_bar[k,j] += A[i,k] * ps_bar[k]
```

And then we see that $\mathrm{ps}_{\mathrm{bar}}$ can be eliminated completely

```
forall i = 0 .. N-1 do
  forall j = 0 .. N-1 do
    forall k = 0 .. N-1 do
      let A_bar[i,k] += B[k,j] * C[i,j]
      let B_bar[k,j] += A[i,k] * C[i,j]
```

We perform some loop distribution and interchange

```
forall i = 0 .. N-1 do
  forall k = 0 .. N-1 do
    let acc = 0.0
    forall j = 0 .. N-1 do
      acc += B[k,j] * C[i,j]
    A_bar[i,k] = acc

forall j = 0 .. N-1 do
  forall k = 0 .. N-1 do
    let acc = 0.0
    forall i = 0 .. N-1 do
      acc += A_T[k,i] * C[i,j]
    let B_bar[k,j] = acc
```

And finally we are ready to rewrite this as a series of maps Which can be optimised

I start with the first $\mathrm{A}_{\mathrm{bar}}$ calculation, going through the following steps, starting with the innermost loop.[1]

```
forall i = 0 .. N-1 do
  forall k = 0 .. N-1 do
    let acc = 0.0
    forall j = 0 .. N-1 do
      acc += B[k,j] * C[i,j]
    A_bar[i,k] = acc
```

[1] While rules were on the slides, i didn't understand till i went through these steps so i wanted to include them

```
forall i = 0 .. N-1 do
  forall k = 0 .. N-1 do
    map2 (*) B[k] C[i] |> reduce (+) 0.0

forall i = 0 .. N-1 do
  map (B_row ->
    map2 (*) B_row C[i] |> reduce (+) 0.0
  )
map (C_row ->
  map (B_row ->
    map2 (*) B_row C_row |> reduce (+) 0.0
  ) B
) C
```

I then perform a similar transformation for the values of `B_bar`, while remembering to transpose the right variables. I interchange the two outer loops, since the indexing of B and A are reversed.

```
let B_bar =
map (\A_T_row ->
  map (\C_t_r ->
    map2 (*) A_T[k] C_t_r
    ) (transpose C)
  ) (transpose A)
```

The final fully optimized code is:

```
let A_bar =
map (C_row ->
  map (B_row ->
    map2 (*) B_row C_row |> reduce (+) 0.0
  ) B
) C
let B_bar =
map (\A_T_row ->
  map (\C_t_r ->
    map2 (*) A_T[k] C_t_r
    ) (transpose C)
  ) (transpose A)
```