DPP Weekly Assignment 1

Simon Vinding Brodersen(rzn875)

November 26, 2025

Note on benchmarking

Throughout this assignment all benchmarking was done using the Hendrix cluster with the NVIDIA GeForce GTX Titax X GPU. Further all testing was done using the builtin *futhark bench* command. I don't know the specifics of which CPU the batch jobs were run on, but if we assume these are the same as the default ssh machine, then it is: Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz.

Further, whenever there is a speedup graph line, this is in reference to the baseline Sequential C runtime.

1 Getting started

1.1 Write a function

The program is a straight forward map-reduce composition.

```
def process [n] (xs: [n]i32) (ys: [n]i32) : i32 =
  reduce i32.max 0 (map i32.abs (map2 (-) xs ys))
```

And is tested via the following test cases:

```
-- Process tests
-- ==
-- entry: test_process
-- nobench input { [23 ,45 , -23 ,44 ,23 ,54 ,23 ,12 ,34 ,54 ,7 ,2 , 4 ,67]
--                 [ -2 , 3 , 4 ,57 ,34 , 2 , 5 ,56 ,56 , 3 ,3 ,5 ,77 ,89] }
-- output { 73 }
-- nobench input { empty([0]i32) empty([0]i32) }
-- output { 0 }
entry test_process = process
```

Which seems to work.

1.2 Benchmark your function

The benchmarking can be seen in Figure 1. The runtimes are reported via a log scale, where we can see that initially the GPU does not offer better performance. However, as the size of the input increases the GPU scales better and overtakes the sequential and multicore runtime.

1.3 Extend your function

The definition of the test_process_idx is provided below

```
def process_idx [n] (xs: [n]i32) (ys: [n]i32) : (i32, i64) =
  let max (d1, i1) (d2, i2) =
      if d1 > d2 then (d1, i1)
      else if  d1 < d2 then (d2, i2)
```

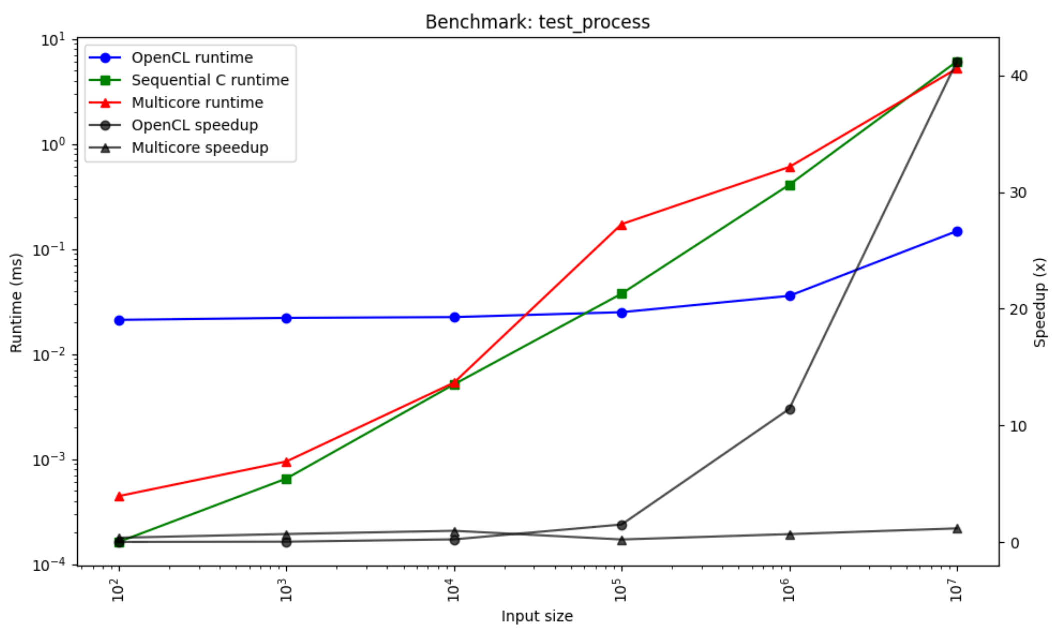

Figure 1: Runtime and speedup of the program written in Section 1.1

```
        else if i1 > i2 then (d1, i1)
        else (d2, i2)
     in reduce_comm max (0, -1)
        (zip
           (map i32.abs (map2 (-) xs ys))
           (iota n)
        )
```

The function implements a max function which is cumulative by using the indices for tie-breaking. This lets us use the faster reduce_comm command. The default reduce function optimises this already when we use a built in operator which is cumulative.

The function is then tested and benchmarked as follows:

```
-- Process idx tests
-- ==
-- entry: test_process_idx
-- nobench input { [23 ,45 , -23 ,44 ,23 ,54 ,23 ,12 ,34 ,54 ,7 ,2 , 4 ,67]
--                 [ -2 , 3 , 4 ,57 ,34 , 2 , 5 ,56 ,56 , 3 ,3 ,5 ,77 ,89] }
-- output { 73 12i64 }
-- nobench input { empty([0]i32) empty([0]i32) }
-- output { 0 -1i64 }
-- notest random input { [100]i32 [100]i32 }
-- notest random input { [1000]i32 [1000]i32 }
-- notest random input { [10000]i32 [10000]i32 }
-- notest random input { [100000]i32 [100000]i32 }
-- notest random input { [1000000]i32 [1000000]i32 }
-- notest random input { [10000000]i32 [10000000]i32 }
entry test_process_idx = process_idx
```

And this seems to work. The benchmarking can be seen in Figure 2, which provides the same story as in the previous section.

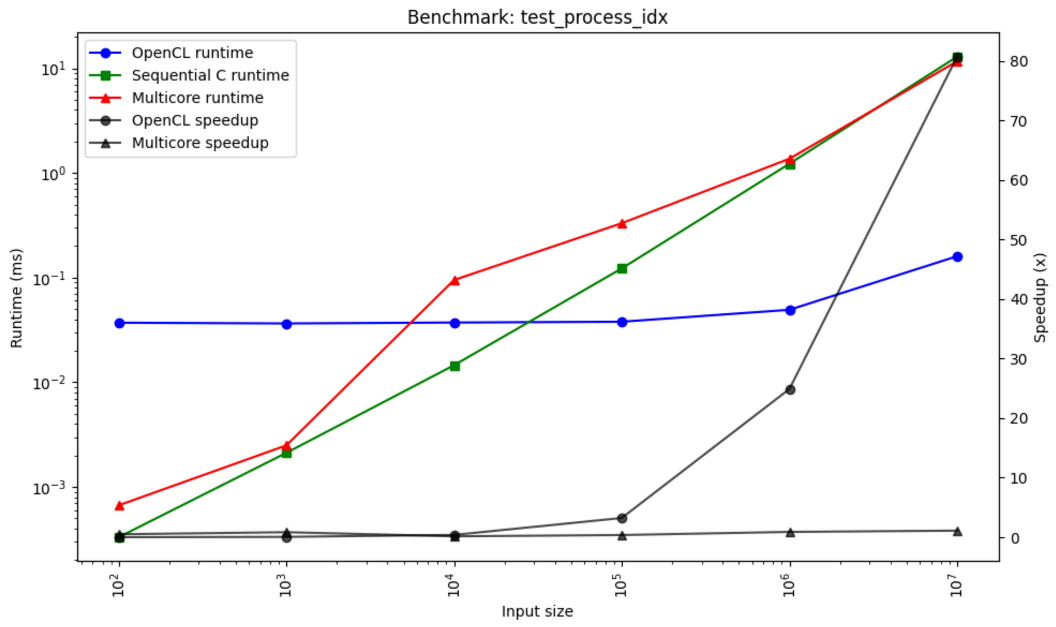

Figure 2: Runtime and speedup of the program written for Section 1.3

2 Implementing prefix sum

The hillis steele implementation can be seen below:

```
def hillis_steele [n] (xs: [n]i32) : [n]i32 =
  let m = ilog2 n
  in loop xs = copy xs for d in 0...m-1 do
    let pow_2 = 1 << d
    in map (\i -> if i - pow_2 >= 0 then xs[i-(pow_2)] + xs[i] else xs[i]) (iota n)
```

This works quite intuitively as we are assuming our input is some power of 2. First, we compute m via the provided ilog2 function. Then, in a loop given m we map over all the indices. If an index is less than the current 2^d, then we simply copy the previous iteration result and otherwise we add $xs[i - pow\_2] + xs[i]$.

This was tested as follows:

```
-- Hillis test
-- ==
-- entry: test_hillis
-- nobench input { [0, 0, 1, 0, 0, 0, 0, 0] }
-- output { [0, 0, 1, 1, 1, 1, 1, 1] }
-- nobench input { [0, 1, 1, 0, 0, 0, 0, 0] }
-- output { [0, 1, 2, 2, 2, 2, 2, 2] }
-- nobench input { [3, 1, 7, 0, 4, 1, 6, 3] }
-- output { [3, 4, 11, 11, 15, 16, 22, 25] }
entry test_hillis = hillis_steele
```

And passes the provided tests.