Weekly Assignment 1 - Report

Francesco Done'
qwg586@alumni.ku.dk

September 10, 2020

Contents

1 Task 1

1.1 Task 1.a

1.1.1 Associativity

Let h as

$$h : A \to B \tag{1}$$

and *Img(h)* as

$$Img(h) = \{h(a) \mid \forall a \in A\} \tag{2}$$

From (2) we know that

$$\{x, y, z\} = \{h(a), h(b), h(c)\} \in Img(h) \tag{3}$$

It is legal to write

$$a + +(b + +c) = (a + +b) + +c$$

because they are lists, therefore with (3)

$$h(a + +(b + +c)) = h(a) \; o \; h(b + +c) = h(a) \; o \; (h(b) \; o \; h(c)) = x \; o \; (y \; o \; z)$$

and

$$h((a + +b) + +c) = h(a + +b) \; o \; h(c) = (h(a) \; o \; h(b)) \; o \; h(c) = (x \; o \; y) \; o \; z$$

so finally we have that

$$x \; o \; (y \; o \; z) = (x \; o \; y) \; o \; z$$

1.1.2 Neutral element e

We know that

$$\forall b \in Img(h) \; \exists a \in A \mid h(a) = b \tag{4}$$

so $h(a)$ can be written as

$$h([\;\;] + +a) = (h([\;\;]) \; o \; h([a])) \tag{5}$$

By definition, we have that

$$h([\;\;]) = e \tag{6}$$

and

$$h([a]) = f(a) = b \tag{7}$$

so, with (5), (6) and (7)

$$h([\;\;] + +a) = (h([\;\;]) \; o \; h([a])) = (e \; o \; b)$$

We have now a sublist of B formed by $\{[\;], \; b\}$ that is equal to $\{b\}$ and equal to $\{b, \; [\;]\}$, so by associativity [1.1.1] is also legal that e o $b = b$ o $e = b$.

1.2 Task 1.b

We have that

$$(reduce(+)0)\ o\ (mapf)\ o\ (reduce(++)[\ \ ])\ o\ distr_p \tag{8}$$

and we know that

$$(reduce(++)[\ \ ])\ o\ distr_p == e \tag{9}$$

(it results in the identity function, which is the neutral element for function composition) after some substitution, it can be written that

$$(reduce(+)0)\ o\ (mapf)\ o\ (reduce(++)[\ \ ])\ o\ distr_p == (reduce(+)0)\ o\ (mapf)\ o\ e$$

applying the third lemma (lecture notes page 17)

$$(reduce(+)0)\ o\ (map(reduce(+)0))\ o\ (mapf)\ o\ distr_p == (reduce(+)0)\ o\ (mapf)$$

finally applying the first lemma (lecture notes page 17)

$$(reduce(+)0)\ o\ (map((reduce(+)0)\ o\ f)\ o\ distr_p == (reduce(+)0)\ o\ (mapf)$$

2 Task 2

Here below there is the code implemented in `lssp.fut`:

```
-- lssp.fut
-- ...
let connect= if tlx == 0 || tly==0 then true else pred2 lastx firsty
let newlss = if connect then max(max((lcsx+lisy), lssx), lssy) else max(lssx, lssy)
let newlis = if connect && tlx == lisx then tlx+lisy else lisx
let newlcs = if connect && tly == lcsy then tly+lcsx else lcsy
let newtl = tlx+tly
-- ...
```

2.0.1 Datasets

There were used three different datasets:

- `lssp-same.fut`: a random dataset with at least some "same-numbers" in a row (e.g. -7).

```
-- compiled input {
-- [2, 42, 19, 54, 40, 56, 24, 12, 47, 5, -2, 38, 72, -7, -7, -7, -7, 23,
    30, 50, 10, 65, 28, 36, 20, 46, 43, 62, 16, 71, 0, -9, 59, 59, 59, 57,
    1, 37, 33, 14, 49, 21, -1, 3, 32, -4, -5, 79, -3, 55, 26, 52, 78, 17,
    6, 64, 61, 41, 44, 7, 51, 70, 60, 9, 76, 22, 63, 31, 34, 58, 69, 48,
    75, 73, -6, 29, 77, 74, 67, 27, 68, 15, 80, 45, 53, 18, 13, 4, 11, 35,
    39]
-- }
-- output { 4 }
```

- `lssp-sorted.fut`: a completely random dataset.

```
-- compiled input {
-- [2, 42, 19, 54, 40, 56, 24, 12, 47, 5, -2, 38, 72, -7, 8, 25, 66, 23,
    30, 50, 10, 65, 28, 36, 20, 46, 43, 62, 16, 71, 0, -9, 59, -8, -10, 57,
    1, 37, 33, 14, 49, 21, -1, 3, 32, -4, -5, 79, -3, 55, 26, 52, 78, 17,
    6, 64, 61, 41, 44, 7, 51, 70, 60, 9, 76, 22, 63, 31, 34, 58, 69, 48,
    75, 73, -6, 29, 77, 74, 67, 27, 68, 15, 80, 45, 53, 18, 13, 4, 11, 35,
    39]
-- }
-- output { 4 }
```

- `lssp-zeros.fut`: a random dataset with at least some zeros in a row.

```
-- compiled input {
-- [2, 42, 19, 54, 40, 56, 24, 0, 0, 0, 0, 38, 72, -7, 8, 25, 66, 23, 30,
    50, 10, 65, 28, 36, 20, 46, 43, 62, 16, 71, 0, -9, 59, 0, 0, 0, 0, 0,
    0, 14, 49, 21, -1, 3, 32, 0, 0, 79, -3, 55, 26, 52, 78, 17, 6, 64, 61,
    41, 44, 7, 51, 70, 60, 9, 76, 22, 63, 31, 34, 58, 69, 48, 75, 73, -6,
    29, 77, 74, 67, 27, 68, 15, 80, 45, 53, 18, 13, 4, 11, 35, 39]
-- }
-- output { 6 }
```

2.0.2 Performance

Using a huge dataset as `futhark dataset --i32-bounds=-10:10 -b -g [10000000]i32 | ./filename -t /dev/stderr -r 10`, the average results were:

- `lssp-same.fut`:

  c: 23931

  opencl: 721

  Gain: +96.987%

- `lssp-sorted.fut`:

  c: 23848

  opencl: 740

  Gain: +96.897%

- `lssp-zeros.fut`:

  c: 12824

  opencl: 742

  Gain: +94.214%

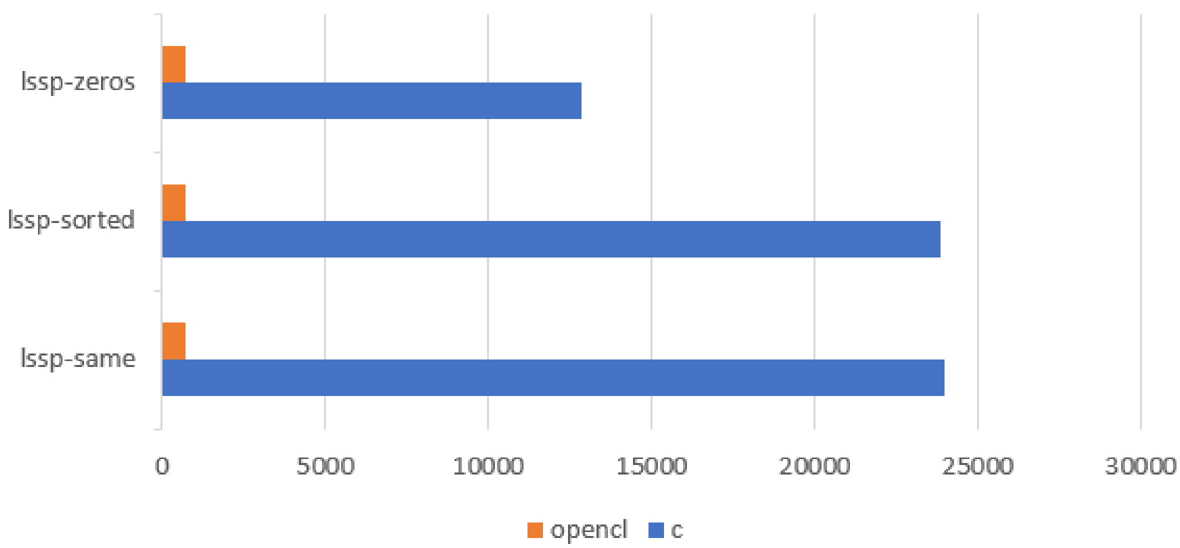

Figure 1: Performances on different backends

3 Task 3

3.1 Validation

The progam validates all the outputs since `h_out[i]==h_out_cpu[i]` $\forall\, i$ from 0 up to $N-1$, where the array `h_out` is performed by GPU and `h_out_cpu` by the CPU.

```
for(unsigned int i=0; i<N; ++i){
   if (fabs(h_out[i] - h_out_cpu[i]) >= 0.0001){
      printf("INVALID at position %d\n",i);
   }
}
```

3.2 Speedups

There were used different array sizes, in particular {1, 8, 32, 54, 128, 256, 512, 1024, 753411} (X axis denotes the size of the array). As you can see in the chart, the GPU-time of usage compared with the one of the CPU, is like 1% with a huge array, and with a tiny array the GPU is 80% faster than the CPU (in the worst case!).

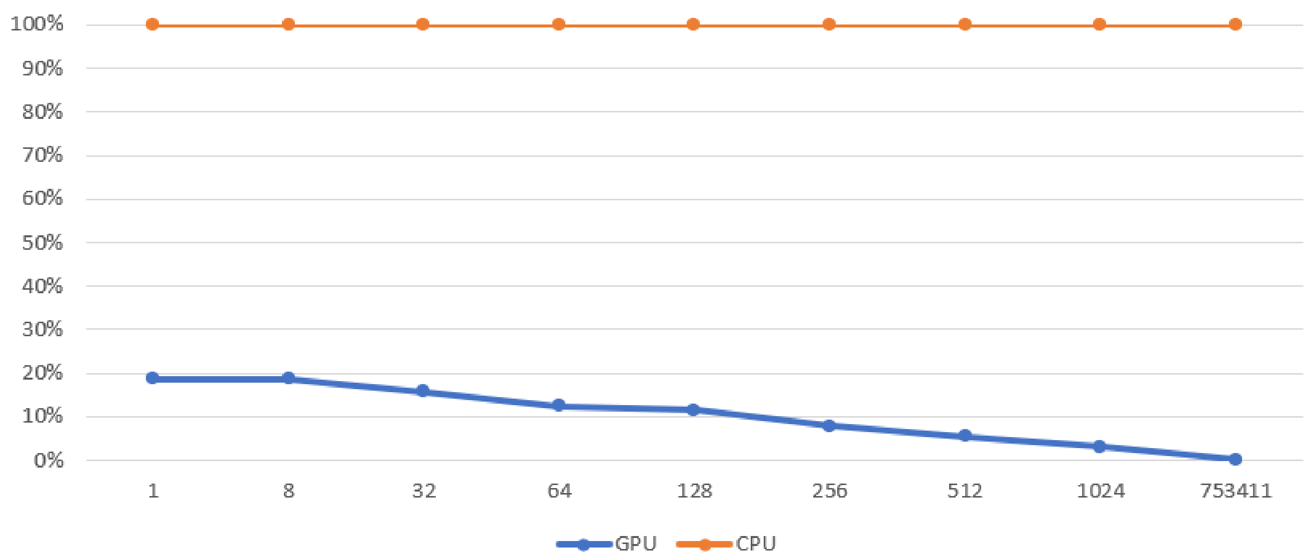

Figure 2: GPU vs CPU speedups with different array sizes

3.3 Code

```
//wa1-task3.cu
//...
__global__ void squareKernel(float* d_in, float *d_out, int sizeN) {
   const unsigned int lid = threadIdx.x; // local id inside a block
   const unsigned int gid = blockIdx.x*blockDim.x + lid; // global id
   if(gid<sizeN){
      d_out[gid] = powf((d_in[gid]/(d_in[gid]-2.3)), 3); // do computation
   }
```

```
}
//...
unsigned int block_size = 256;
unsigned int num_blocks = ((N + (block_size - 1)) / block_size);
//...
// execute the kernel
for(int i=0; i<GPU_RUNS; i++) {
   squareKernel<<< num_blocks, block_size>>>(d_in, d_out, N);
}
cudaThreadSynchronize();
//...
```

4 Task 4

```
-- spMVmult-flat.fut
-- ...
-- 1) compute the flag array from shp
let shp_sc = scan (+) 0 mat_shp
let size = (last shp_sc) + (last mat_shp)
let flags = scatter (replicate size 0) shp_sc mat_shp
let tmp = replicate size 1
let iots = sgmSumF32 flags tmp
in replicate num_rows 0.0f32
-- 2) multiply all elements of the matrix with their corresponding vector element
-- 3) sum up the products above across each row of the matrix. This can be achieved
    with a segmented scan and then with a map that extracts the last element of the
    segment.
```