Exercise Sheet 1
Parallel Functional Programming (PFP) 2017/2018
Version 1.00

Department of Computer Science, University of Copenhagen

2017-11-24

Introduction

These exercises aim at exercising the use of Futhark for writing parallel programs in a functional setting. The exercises assume access to a computer with Futhark installed. For information about installing Futhark, please consult `https://futhark-lang.org`. Points are given according to the following table:

Exercise	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	2.5	3.1	3.2	3.3	3.4	Total
Points	10	10	10	10	10	10	10	??	??	10	10	10	??	100

1 Computing on Signals

This exercise aims at illustrating how simple parallel problems can be expressed in Futhark.

Exercise 1.1 10 P.

Create a Futhark function called `process` that takes as arguments two one-dimensional `i32` arrays (signals) of the same length and computes the maximum absolute difference (pointwise) between the signals (you should not use Futhark's `loop` construct). The function should return the value 0 if two empty signals are passed to the function.

Consider the following two signals:

```
let s1 = [23,45,-23,44,23,54,23,12,34,54,7,2, 4,67]
let s2 = [-2, 3,  4,57,34, 2, 5,56,56, 3,3,5,77,89]
```

What is the result of calling your function on `s1` and `s2`?

Hint: You need to create a `main` function that passes the two arrays to the function `process`.

Exercise 1.2 10 P.

Use the `futhark-dataset` tool to generate seven sets of test data of different length. Each set should contain a pair of one-dimensional `i32` arrays each containing integers in the range [-10000;10000]. The array lengths for the seven different sets should be 100, 1000, 10000, 100000, 1000000, 5000000, and 10000000.

Run the function `process` with the different data sets and with executables obtained both with using `futhark-c` and `futhark-opencl`. Map the timings (in microseconds) onto a chart and remember to specify the system on which you're running the executables.

Exercise 1.3 10 P.

Create a refined version of the `process` function, called `process_idx`, that also returns the index of the source signals for which the maximum absolute difference is found.

Report the result of calling `process2` on the signals `s1` and `s2`. Show evidence that your solution scales as the `process` function.

Hint: The lecture slides should give you a hint to solving this problem.

Exercise 1.4 10 P.

Assuming $\oplus$ is an associative operator with neutral element `0`, show that `(0,false)` is a left-neutral element of

$$(v_1, f_1) \oplus' (v_2, f_2) = (\texttt{if}\ f_2\ \texttt{then}\ v_2\ \texttt{else}\ v_1 \oplus v_2,\ f_1 \vee f_2)$$

2 Transforming Vector Images

In this exercise, you're asked to use the line plotting functions from the lecture slides as a basis for vizualizing translation and rotations of lines.

Code for showing an image is available in the file
`https://github.com/HIPERFIT/futhark-book/blob/master/src/lines_seq.fut`

Exercise 2.1 10 P.

Write a function `transl_point` that takes a translation offset (*xoff*, *yoff*) and a point (x, y) as arguments and translates the point according to the translation offset. Write a function `transl_img` for translating an entire image (an array of lines) according to a given translation offset.

Exercise 2.2 10 P.

Write a function `rotate_point` that rotates a point a number of radians around the origin $(0, 0)$. In math notation, the formula for finding the rotated coordinates (x', y') for a point (x, y), given a number of radians f is $(x', y') = (x \cos f - y \sin f, y \cos f + x \sin f)$.

Exercise 2.3 10 P.

Using the previous definitions, write a function `rotate_img` that rotates an image around its center, where the size of the image is specified in the constants `height` and `width`. The function can assume that all lines are completely within the image boundaries—also after rotation.

Test your function on the image from the lecture slides.[1]

★ **Bonus Exercise 2.4**

Change the function so that lines that cross the image boundaries are correctly clipped.

★ **Bonus Exercise 2.5**

Using one of Futhark's visualisation demo programs, such as `nbody`,[2] as a template, write a program that uses the line drawing routines for drawing and rotating a scene made of lines based on keyboard input.

3 Monte Carlo Simulation

In this exercise, we shall use the technique of Monte Carlo simulation for computing the value of π. We shall first use a simple technique based on an

[1] The `viz.sh` function is available from `https://github.com/HIPERFIT/futhark-book/blob/master/src/viz.sh`.

[2] See `https://github.com/HIPERFIT/futhark-benchmarks/tree/master/accelerate/nbody`.

external generation of random numbers. We shall then use the concept of Sobol-numbers for approaching the real value of π with less work.

Exercise 3.1 10 P.

In this exercise we shall make use of the "dart-throwing" technique. Observe that if one randomly throws a dart on a square of size 2×2 then the chance of hitting within the enclosed circle of radius 1, provided one hits the square, is $\frac{\pi}{4}$. It is quite easy to determine, using Pythagoras's theorem, whether a throw (x, y) is successful, which it is if its distance to the center of the circle is less than or equal to 1, that is, if $(x-1)^2 + (y-1)^2 \leq 1$. Write a function `estimate_pi` that takes two arrays of `f32` values as arguments, corresponding to x and y coordinates for the throws, and, based on the above observations, gives an estimate on the value of π.

Exercise 3.2 10 P.

Using the `futhark-dataset` tool, generate three datasets of different sizes, each containing two arrays of type `[]f32` of the same size, containing `f32` values between 0.0 and 2.0. Use 100, 10000, and 1000000 as the sizes for the data sets. Both for executables generated with `futhark-c` and `futhark-opencl`, plot the time for computing π as a function of the different data set sizes. What do you see?

Hint: Use the following command to generate input for the 100-size case:

```
futhark-dataset --f32-bounds='0:2' -g [100]f32 -g [100]f32 > pi100.inp
```

Exercise 3.3 10 P.

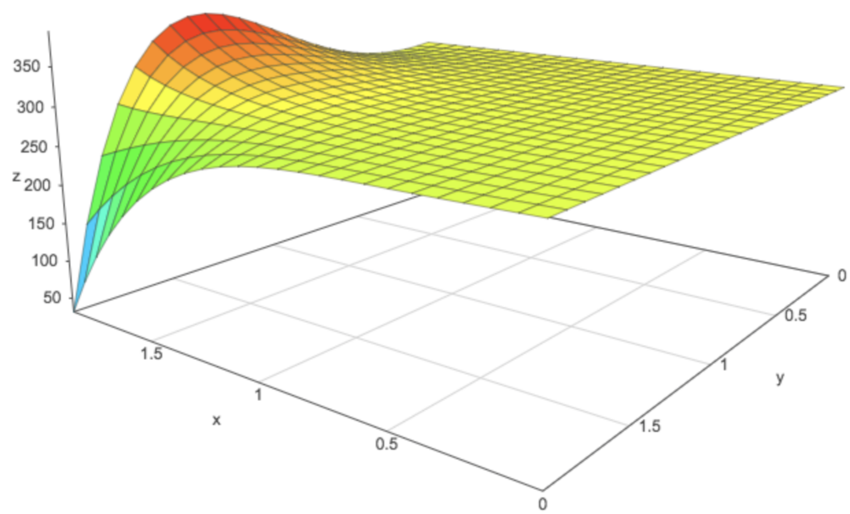
Use the same technique as above but for integrating (i.e., finding the volume under) the following mathematical function (a function of two variables) in the interval $x \in [0; 2]$ and $y \in [0; 2]$:

$$f(x, y) = 2x^6y^2 - x^6y + 3x^3y^3 - x^2y^3 + x^3y - 3xy^2 + xy - 5y + 2x^5y^4 - 2x^5y^5 + 250$$

A Futhark function resembling the mathematical function is given as follows:

```
let f(x:f32) (y:f32) : f32 =
  2.0f32*x*x*x*x*x*x*y*y - x*x*x*x*x*x*y
  + 3.0f32*x*x*x*y*y*y - x*x*y*y*y +
  x*x*x*y - 3.0f32*x*y*y + x*y -
  5.0f32*y + 2.0f32*x*x*x*x*x*y*y*y*y -
  2.0f32*x*x*x*x*x*y*y*y*y*y + 250.0f32
```

Here is a graph showing the surface defined by the function in the interval:



Both for executables generated with `futhark-c` and `futhark-opencl`, plot the time for computing the integral as a function of the different data set sizes for the data sets generated in the previous exercise. What do you see now?

Hint: If n is the number of sample pairs S, the integral $\int_0^2 \int_0^2 f(x, y) dx dy$ can be approximated by $\frac{4}{n} \Sigma_{(x,y) \in S} f(x, y)$.

★ Bonus Exercise 3.4

Futhark features a standard library, which includes a module for generating so-called *Sobol* sequences, an example of quasi-random low-discrepancy sequences. Such sequences are particularly good for Monte-Carlo techniques in that results converge faster than if ordinary pseudo-random numbers are used.

The Futhark module `/futlib/sobol` includes a number of (higher-order) sub-modules, which can be composed to setup a module for implementing a Monte-Carlo simulation. An example use of the library, for establishing the value of π, is available in the file `https://github.com/diku-dk/futhark/blob/master/tests/futlib_tests/sobol.fut`.

Use the sobol library for establishing a value for the integral in Exercise 3.3 and investigate how fast (as a function of the number of sampling points) the integral value converges compared to using the technique presented in Exercise 3.2.