Weekly Assignment 3
Parallel Functional Programming

Troels Henriksen and Cosmin Oancea
DIKU, University of Copenhagen

December 2023

Introduction

The handin deadline is the 15th of December.

The handin is expected to consist of a report in either plain text or PDF file (the latter is recommended unless you know how to perform sensible line wrapping) of 4—5 pages, excluding any figures, along with an archive containing your source code. The report should contain instructions on how to run and benchmark your code.

Task 1: Flattening an If-Then-Else Nested Inside a Map

Please read the Rule 8 of Flattening at slides $39-40$ (or $62-64$) of lecture `L5-irreg-flattening.pdf`.

Your task is to implement the `flatIf` function in file `flat-if-then-else.fut`, such that `flatIf` is the (flat-parallel) code resulted from flattening the following program:

```
map (\b xs -> if b then map f xs
                   else map g xs
    ) bs xss
```

Please note that in the lecture slides `f` is `f xs = map (+1) xs`, but in the example above it would be `f x = x + 1`.

Also, please be mindful that the slide shows a pseudocode that is not (entirely) correct Futhark code—othwerwise it would not fit on one slide. In particular:

- If you know that the flat-data array has size n, and want to create a corresponding flag array (of `i32`), then you would probably want to

dynamically cast the type result obtained from calling *mkFlagArray* to `[n]i32`, e.g., `let flags = (mkFlagArray ...) :> [n]i32`

- the two calls to `split` in the slides are also illegal in Futhark: assuming an array `A: [n]i32`, a call such as `split brk A` should be re-written as `split (A :> [brk + (n-brk)]i32)`

Please see the comments in `flat-if-then-else.fut`.
Once the provided dataset validates, you should:

- make at least one other smallish dataset and check that it validates (size of the array should be in tens-of-elements range)
- create a large dataset (tens-of-million elements range) and report the speedup with respect to a sequential implementation. The latter could be the one compiled with `futhark c` or you may implement an optimized sequential one in a different program – your choice.

Task 2: Flattening Rule for Scatter inside Map (Pen and Paper)

The lecture slides `L5-irreg-flattening.pdf` have presented many flattening rules, but there is none that handles a segmented scatter, i.e., a `scatter` nested inside a `map`.

This was intentionally left for you to implement: your task is to write a rewrite rule for the code below (that of course produces flat-parallel code):

```
map (\xs is vs -> scatter xs is vs
    ) xss iss vss
```

This is a pen and paper exercise, so describe it in your report. You may assume that `xss`, `iss` and `vss` are two-dimensional irregular arrays whose shape and flat data representation are known. (From the semantics of `scatter` it also follows that the shape of `iss` is the same as the shape of `vss` but may be different than the shape of `xss`).

You will probably have to use this rule in the implementation of `partition2` lifted.

Task 3: Flattening Rule for Histogram inside Map (Pen and Paper; very similar with the previous task)

The lecture slides `L5-irreg-flattening.pdf` have presented many flattening rules, but there is none that handles a segmented reduce-by-index, i.e., a `reduce-by-index` nested inside a `map`.

This was intentionally left for you to implement: your task is to write a rewrite rule for the code below (that of course produces flat-parallel code):

```
map (\histo is vs -> reduce_by_index (⊙) 0⊙ histo is vs
    ) histos iss vss
```

This is a pen and paper exercise, so describe it in your report. You may assume that `histos`, `iss` and `vss` are two-dimensional irregular arrays whose shape and flat data representation are known. (From the semantics of `reduce_by_index` it also follows that the shape of `iss` is the same as the shape of `vss` but may be different than the shape of `histos`).

Task 4: Implement the lifted version of `partition2`

The file `quicksort-flat.fut` implements the flat-parallel code for quicksort, but the implementation is incomplete.

Your task is to implement function `partition2L`, which is the lifted version of function `partition2` (provided).

Please see the comments in file `quicksort-flat.fut` and the relevant slides from `L5-irreg-flattening.pdf` (pertaining "Flattening Quicksort" section towards the end). To Do:

- Once the implementation of quicksort validates on the small dataset provided in `quicksort-flat.fut`, run the program on a large dataset, e.g., ten-hundreed millions floats, and report runtime and speedup versus the sequential version (e.g., `futhark cuda` *vs* `futhark c`).
- Show your implementation of `partition2L` in your report and describe
  - for each line in `partition2`, what rule have been used to flatten, and what is the corresponding code in `partition2L`.

Please note that, if you have a bug in your code, then it is likely that the futhark program will run forever because the outer sequential loop (of quicksort) ends only when the array gets sorted (and if you have a bug it might never be).