Bfast: Translating Futhark Code Efficiently in CUDA

Preamble

This project refers to writing an efficient CUDA implementation for a variation of the BFAST algorithm, which is one of the state-of-the-art methods for break detection given satellite image time series. For example BFAST can be used to detect environmental changes, such as deforestization in the Amazonian jungle, etc.

The attached paper "Massively-Parallel Break Detection for Satellite Data" reports an implementation that assumes that that all pixels hold valid data. This is somewhat unrealistic because a certain area may be covered by clouds—the tested dataset is from an area located in Sahara, which is never clouded.

The attached Futhark implementation is work in progress (hence do not distribute) and fixes the cloud issue by semantically “filtering” out the clouded pixels—encoded by NAN values–in the implementation of various (mathematical) operations. Note that filtering does not necessarily mean in this context the application of the `filter` operator, but rather it means that the iteration space is padded and `NAN` values are ignored.

For example, the Futhark implementation of dotproduct is:

```
let dotprod [n] (xs: [n]f32) (ys: [n]f32): f32 =
    reduce (+) 0.0 (map2 (*) xs ys)
```

and the filtered implementation receives an additional flag array in which the invalid pixels, marked with the flag-value `false`, are ignored:

```
let dotprod_filt [n] (flgs: [n]bool) (xs: [n]f32) (ys: [n]f32) : f32 =
    f32.sum (map3 (\ flg x y -> if flg then x*y else 0) flgs xs ys)
```

1 Tasks

Your task is to write a CUDA program, which is semantically equivalent with the Futhark one, and which is as efficient as possible:

- Please provide built-in validation for the provided Sahara dataset—files `sahara.in` and `sahara.out` are the reference input and result.
- Identify the opportunities for optimizing locality of reference, and apply related optimizations whenever is possible and profitable to do so, for example:
  1 make sure that all accesses to global memory are coalesced (i.e., spatial locality),
  2 identify the opportunities for one/two/three dimensional tiling in shared memory and apply them whenever profitable;
  3 identify opportunities in which a kernel loop can be executed entirely (or mostly) in shared memory, for example in the case of matrix inversion (e.g., the `gauss_jordan` Futhark function).
  4 identify the cases in which a composition of `scan`, `filter`, `map` or `scatter` operations can be performed in (fast) shared memory. For example, if the size of the scanned segment is less than 1024, it is much faster to perform the `scan` within each block, then to perform a segmented scan across all elements in all blocks (why?).
- Try to report the impact of your optimizations both locally and globally.

  - Locally means to separately test the impact of optimizations: for example what is the speedup of tiling matrix-vector multiplication, with respect to its untiled version;
  - Globally means to test the speedup generated by an optimization (e.g., tiling) at the level of the whole application runtime.

- Also please report your global performance, for example in terms of percentage of the peak bandwidth of the system (and perhaps also in terms of the sequential program — could be the one generated by futhark-c).

2 Closing Remarks

- Please write a tidy report that puts emphasis on the parallelization and optimization strategy reasoning, and which explores and explains the design space of potential solutions, and argues the strengths and shortcomings of the one you have chosen. For example present on representative example how optimizations such as tiling were applied, and why their application is safe.

- Please make sure your code validates at least on the provided datasets, and please compare the performance of your code against baseline implementations in Futhark (or from elsewhere if you have other), and comment on the potential differences between the observed performance and what you assumed it will be (based on the design-space exploration).