Implementation of Fast Sorting Algorithms for GPU

Your task for this project is to implement as fast a sorting algorithm as you can on GPU.

1. For this, you will need to do first a literature search, for example, to understand and explain the parallel implementation of radix-sort (and perhaps merge sort) and to discuss how can it (they) be implemented from parallel basic blocks, such as map, reduce, scan, scatter, and what is the work and depth of such implementation.
2. Please note, that you do not have to implement them yourselves, for example because they are already implemented in Futhark package “github.com/diku-dk/sorts”. You may get a package by the following simple command:

   ```
   $ futhark pkg add github.com/diku-dk/sorts
   ```

   followed by

   ```
   $ futhark pkg sync
   ```

   You can then write some short Futhark programs---which simply call the library---that will serve as a baseline for comparison with your low-level CUDA implementation, which should be significantly faster.
3. The third step is to do a literature search to find what sorting algorithms have been found to be most suited for GPU execution. You will find such a paper saved in the corresponding folder of this project, and even a slide presentation. If the paper is unclear, you may always come to discuss it with Cosmin, and/or you can run your own literature search. Please note that the selected sort algorithm does not have to be stable or deterministic. It is a good bet that a fast algorithm resembles to some extent radix-sort, and it applies on key-value pairs.
4. Implement in CUDA at least one of the fast sorting algorithms for GPU, and evaluate its performance in comparison with the Futhark baseline and with the implementation provided by the CUB library (written in CUDA). A sample CUDA program that uses the CUB library is also provided in the folder associated with this project. Cosmin has found that on one GPU hardware the CUB library is about 19x faster than Futhark’s radix sort. Hopefully your implementation will be competitive with the one from CUB.
5. Then try to generalize your implementation to work with any datatype, for example, single/double precision int or floats, or even tuples. For this take a look at how this was achieved in the “pbb” library that you have helped implement in weekly-2.
6. Finally, please provide a detailed performance evaluation on multiple datasets of various

datatypes, and at least compare the performance with the ones of the Futhark baselines and with the CUB library.

7. Please write a tidy report in which you:
    1. Start from explaining at least the radix sort algorithm, its assumptions, and its parallel work-depth complexity.
    2. Then move to presenting in detail---including high-level pseudocode and written explanation---the fast algorithm that you have chosen to implement. Reason if possible about its work-depth asymptotic, but also about the reasons due to which you expect to run faster than radix sort.
    3. Present your optimized CUDA implementation---in both pseudocode and text---with emphasis on reasoning about what CUDA features you have used for optimion (e.g., atomic addition?). Also reason about the tradeoffs: For example, is your implementation a stable sort, is it deterministic, is it a true data-parallel implementation (meaning can you write it in Futhark in terms of nested map, scan, reduce, scatter, reduce_by_index)?
    4. Present a systematic performance evaluation in which you compare on several datasets and datatype the performance of your CUDA implementation in comparison with the Futhark baselines and with that of the CUB library.