YSC3248: Parallel, Concurrent and Distributed Programming

Programming Assignment 6

In this assignment, you will be practicing your understanding of the idea of *splitting* in parallel collections. To do so, you will have to implement a familiar binary heap structure and enhance it with the possibility to perfrom non-modifying operations in parallel, by means of splitting.

The template repository for this assignment is available on Canvas. Once you clone the template repository, you'll be ready to go! The files you need to modify are:

- `ParBinHeap.scala`
- `BinHeapTests.scala`
- `ParBinHeapSplitter.scala`
- `ParBinHeapSplitterTests.scala`
- `ParBinHeapBenchmarks.scala`

Start by implementing a textbook version of a correct sequential binary min-heap by developing the corresponding methods of `ParBinHeap`. Whenever necessary, feel free to add your own fields and methods, as well as constructor parameters. Next, test your implementation by implementing and extending the test suite in the `BinHeapTests` class.

For the second part of this assignment, implement a splitting discipline by filling the gaps in the class `ParBinHeapSplitter`. Finally, test that your parallel heap behaves correctly with standard split-based operations (`reduce`, `aggregate`, `fold`, etc), and that they don't affect the structure of the heap. You are also encouraged to develop any additional tests you deem necessary.

To hand in your code for this assignment, submit the link to the tagged release in your repository on GitHub. Feel free to provide additional comments about your implementation and discoveries to the `README.md` file in the root of the project.

Here are some tips to get you started on the assignment:

- You can find an implementation of a binary max-heap in OCaml in the lecture notes for YSC2229.[1]
- The type parameter annotations `T: Ordering: ClassTag` generate the *implicit* constructor parameters of types `Ordering[T]` and `ClassTag[T]` that are passed to the constructors of `ParBinHeap` class. The former one allows one to deduce the priority discipline for the insertions. It will be inferred automatically when working with standard data types, such as `Int` and `String`, but you will need to define it yourself for custom classes, or in the case if the default ordering is not suitable for the problem. The class `BinHeapTests` provides examples on how to do this.
- Notice that the `split` for the binary heap splitter `ParBinHeapSplitter` is different from the one in the class, as it doesn't take a series of chunk sizes. That is, it is up to you how to split the heap and what are the partition that the recursive splitters should contain.
- Don't worry about the iterator's methods (`hasNext`, `next`, and `remaining`) running before `split`. They typically are run on the chunks of the collection *after* all splitting has been performed. That is, splitting and iteration do not interleave.
- Do not worry too much if the parallelism doesn't pay off in terms of actual performance (although we hope it will!). For your experiments, you may want to design your benchmarks by so they would perform some computationally "heavy" operations on elements of the heap.

[1] https://ilyasergey.net/YSC2229/YSC2229-lecture-notes-week-05.html