Assignment 1: Parallel Image Filter

The class **`ImageFilter`** implements a sequential, iterative nine-point image convolution filter working on a linearized (2D) image. In each of the **`NRSTEPS`** (=100) iteration steps, the average RGB-value of each pixel *p* in the source array is computed, considering *p* and its 8 neighbor pixels (in 2D), and written to the destination array.

The program **`TestImageFilter`** can be used to filter a JPG image, e.g., as follows:

```
java TestImageFilter IMAGE1.JPG
```

The resulting filtered output image is: **`FilteredIMAGE1.JPG`**

`IMAGE1.JPG`

`FilteredIMAGE1.JPG`

The assignment on Moodle includes a zip file (Assignment1.zip), which contains the source files **`TestImageFilter.java`**, **`ImageFilter.java`**, and two example images, **`IMAGE1.JPG`** and **`IMAGE2.JPG`**, and a text file **`out.txt`**, which shows how the output of the extended test program might look like.

Parallelization

Implement a class **`ParallelFJImageFilter`** for parallel image filtering using the Java **Fork/Join framework.** For parallelization, the pixels in the image should be recursively partitioned among all the tasks used for parallel execution. Use a proper threshold to limit the number of taks generated. Take care of **proper synchronization** after each iteration!

`ParallelFJImageFilter` must offer

- a public constructor with the same parameter types as class **`ImageFilter`**, and
- a public method **`void apply(int nthreads)`**, where the parameter **`nthreads`** corresponds to the number of threads used for parallel execution.

Extend the provided test program (**`TestImageFilter`**) such that it also invokes the parallel image filter repeatedly using 1, 2, 4, 8, 16 and 32 threads, respectively. For each invocation of the parallel image filter, the elapsed time and the corresponding speedups need to be reported.

The test program has to verify for all parallel image filter executions that the **output image is exactly the same as the one produced by the sequential image filter** (pixel-wise comparison!).

In addition, the test program has to verify for all parallel executions that the **parallel efficiency is at least 0.7**, especially when #threads=#cores.

Ensure that the parallel image filter works with images of arbitrary size.

Lab and moodle arrangements

1. Put your files in Assignment 1 part in **`Moodle`**.
2. Provide two text files, **`out1.txt`** and **`out2.txt`**, containing the output of one successful program run with **`IMAGE1.JPG`** and **`IMAGE2.JPG`**, respectively.
3. Provide a short pdf document (**`forkjoinfilter.pdf`**) with a brief documentation/explanation of your solution (e.g., how did you parallelize it, threshold, number of tasks generated, etc.) and with two speed-up curves for the performance results obtained in **`out1.txt`** and **`out2.txt`**, respectively, and provide a brief discussion of each curve.
4. Upload a zip-file (**`ex1.zip`**) with **`ParallelFJImageFilter.java`**, **`TestImageFilter.java`**, **`forkjoinfilter.pdf`**, **`out1.txt`**, and **`out2.txt`** before the deadline to Moodle.

•Requirements for positive grading

- The classes **`ParallelFJImageFilter`** and **`TestImageFilter`** perform the **required functionality as described above**.
- The parallel executions have a **parallel efficiency of at least 0.7**
- The zip file (**`ex1.zip`**) containing the files described above has been **uploaded to Moodle before the deadline**.