Lab C: Parallel Programming in Java 8

DAT280 - Parallel Functional Programming (Group 17)

Theodor Åstrand – theast@student.chalmers.se – 931109-9114
Rafael Mohlin – mrafael@student.chalmers.se – 931106-0017

Chalmers University of Technology

May 4, 2016

Assignment 3.3

3.3.6

Our sequential version of the palindrome-printing took 151ms, while the parallelized one took 26ms. So there is an obvious difference between the sequential and the parallelized running time.

We misinterpreted the question at first, so we implemented a parallel version of isPalindrome. This was very, very bad.

3.3.10

The frequency of the letter 'e' is 235331. We believe that there is an error in the lab assignment, where it says that the frequency of 'e' is 235,886 since this is the same as the amount of words in the vocabulary.

3.3.12

By using collect and groupingBy the parallel version is slower than the sequential one. This is because groupingBy has to keep the order of the stream and thus can't be evaluated in parallel. When we used groupingByConcurrent, the performance doubled. This is fine because we don't care about the order anyway.

Assignment 3.4

3.4.5

The result is incorrect and is larger than what it should be. If you run the program multiple times the summation results differ every time, this means that you cannot trust using mutable state with generate while doing parallel streams. This is because two threads might access the current divisor and increment them at the same time. If they do this one of the threads used the wrong divisor and the divisor also doesn't get incremented twice, which is why the result is larger.

This can be solved with an AtomicLong and its getAndIncrement function, but then the purpose of trying to parallelize it is defeated. Also when we tried this it took more than twice the amount of time to compute it, which is probably due to the synchronization overhead introduced by the AtomicLong.