miniproject_4.zip

Project Goals & Outcomes

In lectures this week, you learned about phasers (not the Star Trek kind!) and how they can be used to express "fuzzy" (split-phase) barriers and point-to-point synchronization. You also saw how fuzzy barriers and point-to-point synchronization can be used to improve performance, relative to regular barriers. In particular, fuzzy barriers allow parallel programmers to expose more parallelism than a simple barrier (specified by the *arriveAndAwaitAdvance()* API) by overlapping the completion of a barrier with useful work. In a fuzzy barrier, the completion of the barrier is split into two steps. In the first step (the *arrive()* API), each thread signals to other threads that it has reached a point in its own execution where it is safe for any other thread to proceed past the barrier. In the second step (the *awaitAdvance()* API), each thread checks that all other threads have signaled that they have reached step one to be sure it can proceed past the barrier.

In this mini-project you will apply Java's phasers and Java threads to the One-Dimensional Iterative Averaging algorithm that was introduced in Week 3. Before getting started with this mini-project, it is recommended you review the first demo video in Week 4, in which Professor Sarkar walks through an example similar to this project.

Project Setup

Please refer to Mini-Project 0 for a description of the build and testing process used in this course.

Once you have downloaded and unzipped the provided project files using the gray button labeled miniproject_4.zip at the top of this description, you should see the project source code at:

`miniproject_4/src/main/java/edu/coursera/parallel/OneDimAveragingPhaser.java`

and the project tests at

`miniproject_4/src/test/java/edu/coursera/parallel/OneDimAveragingPhaser.java.`

Project Instructions

Your modifications should be done entirely inside of OneDimAveragingPhaser.java. You may not make any changes to the signatures of any public or protected methods inside of OneDimAveragingPhaser, or remove any of them. However, you are free to add any new methods you like. Any changes you make to OneDimAveragingPhaserTest.java will be ignored in the final grading process.

Your main goals for this assignment are listed below. OneDimAveragingPhaser.java also contains helpful TODOs.

1. Based on the provided reference sequential version in OneDimAveragingPhaser.runSequential and the reference parallel version in OneDimAveragingPhaser.runParallelBarrier, implement a parallel version of the one-dimensional iterative averaging algorithm that uses phasers to maximize the overlap between barrier completion and useful work being completed.

Project Evaluation

Your assignment submission should consist of only the OneDimAveragingPhaser.java file that you modified to implement this mini-project. As before, you can upload this file through the assignment page for this mini-project. After that, the Coursera autograder will take over and assess your submission, which includes building your code and running it on one or more tests. Your submission will be evaluated on Coursera's auto-grading system using 2 and 4 CPU cores. Note that the performance observed for tests on your local machine may differ from that on Coursera's auto-grading system, but that you will only be evaluated on the measured performance on Coursera. Also note that for all assignments in this course you are free to resubmit as many times as you like. See the Common Pitfalls page under Resources for more details. Please give it a few minutes to complete the grading. Once it has completed, you should see a score appear in the "Score" column of the "My submission" tab based on the following rubric:

- 100% – correctness and performance of the fuzzy barrier implementation of one-dimensional iterative averaging on an input of 4 * 1024 * 1024 elements using 4 cores

Mark as completed