coursera

miniproject_3.zip

Project Goals & Outcomes

The aim of this project is intended to test your ability to exploit parallelism in loops. The provided code includes a sequential program for multiplying two matrices. Your task is to write a correct parallel version of the program that runs faster than the sequential version.

In lectures this week, you learned about expressing parallelism across loop iterations. Sequential loops are common targets for parallelization for two reasons. First, because loop bodies are often repeated many times, the majority of execution time of sequential programs is often spent inside of one or more loops. Second, because loops express the same computation repeated across many points in the loop iteration space, parallelism arises naturally as long as those iterations are independent of each other. Of course, it is not always legal to parallelize a loop as data races may result.

More specifically, this week covered two ways of expressing loop parallelism in Java: 1) PCDP's forall methods, and 2) parallel streams in standard Java. In this mini-project, you will get hands-on experience with PCDP's forall methods by parallelizing matrix-matrix multiply. Before getting started with this mini-project, it is recommended you review the first demo video in Week 3 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_3.zip at the top of this description, you should see the project source code at:

`miniproject_3/src/main/java/edu/coursera/parallel/MatrixMultiply.java`

and the project tests at

`miniproject_3/src/test/java/edu/coursera/parallel/MatrixMultiplyTest.java`

Project Instructions

Your modifications should be done entirely inside of MatrixMultiply.java. You may not make any changes to the signatures of any public or protected methods inside of MatrixMultiply, or remove any of them. However, you are free to add any new methods you like. Any changes you make to MatrixMultiplyTest.java will be ignored in the final grading process. As in past mini-projects in this course, the provided code is a fully functioning Maven project but you are not required to use Maven if you prefer since we have also provided the necessary JARs to use an alternate approach to build and execute your code

Your main goals for this assignment are as follows (note that MatrixMultiply.java also contains helpful TODOs):

Modify the MatrixMultiply.parMatrixMultiply method to implement matrix multiply in parallel using PCDP's forall or forallChunked methods. This will closely follow the demo by Prof. Sarkar in the first demo video of Week 3. There is one TODO in MatrixMultiply.parMatrixMultiply to help indicate the changes to be made. A parallel implementation of MatrixMultiply.parMatrixMultiply should result in near-linear speedup (i.e. the speedup achieved should be close to the number of cores in your machine).

Project Evaluation

Your assignment submission should consist of only the MatrixMultiply.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:

- 20% – correctness and performance of the parallel matrix multiply implementation on a 512x512 matrix using 2 cores
- 30% – correctness and performance of the parallel matrix multiply implementation on a 512x512 matrix using 4 cores
- 20% – correctness and performance of the parallel matrix multiply implementation on a 768x768 matrix using 2 cores
- 30% – correctness and performance of the parallel matrix multiply implementation on a 768x768 matrix using 4 cores

✓ Complete