Lab 4: Finish Accumulators and ForkJoinPool

Instructor: Vivek Sarkar, Co-Instructor: Shams Imam

Course Wiki: http://comp322.rice.edu

Staff Email: comp322-staff@mailman.rice.edu

Goals for this lab

- Implement a parallel version of NQueens using Finish Accumulators and the cutoff strategy
- Observe both the abstract and real performance of parallel NQueens with the cutoff strategy
- Understand how ForkJoinPool can be used to parallelize programs using Standard Java by implementing parallel NQueens
- Understand how the Cutoff strategy improves performance

1 Setup

As in previous labs, download the `lab_4` project on your machine using one of the following methods. Note: The URL for lab 4 is `https://svn.rice.edu/r/comp322/turnin/S16/NETID/lab_4/` with `NETID` replaced with your net id.

1. Download the project using the IntelliJ support for Subversion (Instructions with Images).
2. Download the project using the command line (Demo Video).
3. If you do not have subversion set up on your machine, you can download the TODO FIX URL lab_2.zip file and manually set up the project on IntelliJ.

Note that if you have the `-javaagent` set up in your IntelliJ run configuration, you can use standard IntelliJ debugging features (e.g. breakpoints) to debug your code.

2 The N-Queens Problem

This week we will revisit the simple N-Queens problem (*i.e.*,how can we place N queens on an N × N chessboard so that no two queens can capture each other?) introduced last week and in the Lecture 7: Finish Accumulators. You will edit the `NQueensHjLib.java` and `NQueensForkJoin.java` files provided in your svn repository for this exercise. There are `TODO`s in these files guiding you on where to place your edits.

1. The lab code already contains a sequential implementation for solving the N-Queens problem. The first goal of this exercise is to create a parallel version to solve the N-Queens problem using HJlib and finish-accumulators, and then observe the abstract metrics for this solution. These changes should go mostly in `NQueensHjLib.nqueensKernel`.
2. The second goal of this exercise is to use the cutoff strategy in your parallel HJlib version to see performance improvements. For a description of the cutoff strategy, see the Lecture 10 slides posted on the course website. These changes should modify your version of `NQueensHjLib.nqueensKernel` from the previous item.

3. The third goal of this exercise is to create a parallel version of NQueens using ForkJoinPool from standard Java and observe the real execution times. We have provided the `ArrayDivide.java`, `ArraySum.java` and `ArraySumFourWay.java` files for examples on how to use the Fork/Join framework in Java. These changes should go in `NQueensForkJoin.compute` and `NQueensForkJoin.findQueens`.

3 Reminders

Abstract metrics were turned on in Lab 2 but were off by default in Lab 1. In the first part of the lab, you will use the HJlib API to optionally turn on abstract metrics. You will need to add two import statements (they are already present in `NQueensHjLib.java`):

- `import edu.rice.hj.runtime.config.HjSystemProperty`
- `import static edu.rice.hj.Module0.abstractMetrics`

To turn '*on*' abstract metrics you need to ensure "`HjSystemProperty.abstractMetrics.setProperty(true)`" is invoked, just before the call to "`launchHabaneroApp()`".

The purpose of the call to `launchHabaneroApp()` is to launch the specified code expression as a lambda to be executed in parallel by the HJlib runtime, while all code before and after the call to `launchHabaneroApp()` is executed as standard Java code. For the current version of HJlib, it is good practice to include a top-level `finish` in the body of `launchHabaneroApp()` *In particular, the current implementation of abstract metrics may not print correct results if a top-level* `finish` *is omitted in* `launchHabaneroApp()`.

4 Demonstrating and submitting in your lab work

For this lab, you will need to demonstrate and submit your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab. They will want to see your files submitted to Subversion in your web browser and the passing unit test on your laptop.
2. Check that all the work for today's lab is in your `lab_4` directory by opening `https://svn.rice.edu/r/comp322/turnin` in your web browser and checking that your changes have appeared.
3. Submit all your changes in the `lab_4` directory using subversion. Remember to explicitly add any new files (e.g. your report or any new Java files) you created into the subversion repository.