Lab 3: Futures

Instructor: Vivek Sarkar

Course wiki: `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`

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

Goals for this lab

- Understand futures
- Explore more functional programming
- One HJ-lib API: `future`
- Use a data race detector (FastTrack on RoadRunner) to debug races in HJlib programs

Lab Projects

Note that there are two Maven projects for this lab. They are located in the following svn repositories:

- `https://svn.rice.edu/r/comp322/turnin/S15/`*NETID*`/lab_3_futures`
- `https://svn.rice.edu/r/comp322/turnin/S15/`*NETID*`/lab_3_datarace`

Please use the subversion command-line client to checkout the project into appropriate directories locally. For example, you can use the following commands from a shell:

```
$ cd ~/comp322
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_3_futures lab_3_futures
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_3_datarace lab_3_datarace
```

For the exercises in section 1, you will use the Maven project named `lab_3_futures`. You will use the Maven project named `lab_3_datarace` for the exercises in section 2.

1 Getting Familiar with Futures

1.1 Example Program for Futures: ArraySum2

In this exercise, you will execute the `ArraySum2Test` and observe the output of the parallel array sum as its size increases by powers of 2, up to 128. Note that output will be generated by a series of JUnit tests, which pass if and only if there is some parallelism achieved. Note the Work, CPL, and Ideal Parallelism results. What happens to each of these as the size of the array increases? Are there any unexpected values for ideal parallelism? Why do you think you see these differences?

1.2 Array Sum Revisited with Variable Execution Times: ArraySum4

In this exercise you will execute the `ArraySum4Test` and observe its outputs (as before, in JUnit tests). The main difference between `ArraySum4.java` and `ArraySum2.java` is that the call to `doWork()` here estimates the cost of an add as the number of significant bits in both operands. Thus, the cost depends on the values being added. Again, observe the values of CPL, Work, and Ideal Parallelism. Are these values different from those in the previous exercise? Given what you know about how work is evaluated in this exercise, why do you think that is?

1.3 Binary Trees using Futures

In this exercise, you are given sequential code for a functional implementation of a binary tree. Your task is to convert it to a correctly executing parallel tree. Once correctly implemented, your code should pass the provided `BinaryTreeTest`. Think back on your previous experience with functional programming, and your knowledge of futures and how they relate to your task.

2 Data Race Detection in HJlib Programs

HJlib programs running under the thread-blocking runtime allow automatic instrumentation of the program to check for data races at runtime using the FastTrack tool. This capability can be enabled by allowing the RoadRunner runtime to execute HJlib programs. In the `lab_3_datarace` Maven project, the `pom.xml` has been configured to enable you to run some HJlib programs under FastTrack. FastTrack runs an instrumented version of your HJlib program and reasons about all potential interfering accesses that are candidates for data races according to the parallelism in the input program. If a race is found, a pair of interfering accesses is printed with source code locations for the accesses.

2.1 Racy Reciprocal Array Sum

We have included a racy version of ReciprocalArraySum.java in the maven project. You can run this version of the program and use FastTrack to detect data races by using the following command:

```
$ mvn clean compile exec:exec -PRacyReciprocalArraySum
```

Running this command should detect data races and output something as follows:

```
 ...
##          Thread: 2
##           Blame: rd_array@edu/rice/comp322/RacyReciprocalArraySum.java:133:38
##           Count: 1    (max: 20)
##     Guard State: [W=(3:C) R=(--:--) CV=[(0:0) (0:0) (2:30) (3:C)]]
##  Current Thread: [tid=2    cv=[(0:472) (1:58) (2:30) (3:B) (0:0)]   epoch=(2:30)]
##           Array: @03[0]
## Previous Access: wr_array@edu/rice/comp322/RacyReciprocalArraySum.java:122:20 by Thread 3 (name = Th
##  Current Access: rd_array@edu/rice/comp322/RacyReciprocalArraySum.java:133:38 by Thread 2 (name = po
 ...
```

Lines 122 and 133 have the following code:

```
...
122: partialSums[0] += computeReciprocal(X, i);
...
133: final double sum = partialSums[0] + partialSums[1];
...
```

Using the information we can deduce that the race is occurring on the access to `partialSums[0]`.

Your goal in this lab is to fix the RacyReciprocalArraySum.java so that fastTrack no longer reports a data race.

2.2 Searching Text Patterns

Your final goal in this lab is to parallelize the `searchPar()` methods in Search1.java and Search2.java. You can run the unit tests (`mvn clean test`) to confirm that your parallel solution work correctly. As an aid

to help you debug your solution, we have also set up Maven configuration so that fasttrack can be used in Search1.java or Search2.java. To debug data races in your solution simply type one of the following commands in your shell:

```
$ mvn clean compile exec:exec -PSearch1
$ mvn clean compile exec:exec -PSearch2
```

Turning in your lab work

For each lab, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab (as in COMP 215). Be prepared to explain the lab at a high level, as well as answer the following questions:
   - What is the trend in the abstract metrics as the size of the array increases?
   - Is there a difference between the abstract metrics between ArraySum2 and ArraySum4? Why do you think there is or is not?
   - What was your strategy in rewriting the provided sequential binary tree as a parallel one? How did functional programming aid you in this pursuit?
2. Check that all the work for today's lab is in the `lab_3` directory. If not, make a copy of any missing files/folders there. It's fine if you include more rather than fewer files — don't worry about cleaning up intermediate/temporary files.
3. Use the turn-in script to submit the `lab_3` directory to your turnin directory as explained in the first handout: *turnin comp322-S15:lab_3*. Note that you should *not* turn in a zip file.

   NOTE: Turnin should work for everyone now. If the turnin command does not work for you, please talk to a TA. As a last resort, you can create and email a `lab_3.zip` *file to comp322-staff@mailman.rice.edu.*