Lab 3: Data Race Detection and Repair

Instructor: Vivek Sarkar

Resource Summary

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

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

Coursera Login: visit `http://rice.coursera.org` and select "Fundamentals of Parallel Programming"

Clear Login: ssh *your-netid*@ssh.clear.rice.edu and then login with your password

NOTE: As with Labs 1 and 2, you have the option of doing today's lab on your laptop computer or on a lab computer. If you use your laptop, the setup should work smoothly if it runs Mac OS or Linux. For Windows, you should ensure that DrHJ is launched on a standard Oracle JDK. If you're having problems using a Windows machine for your work, we recommend that you use a lab machine instead. You should have 24-hour access to the lab with your Rice ID card.

Finally, all commands below are CaSe-SeNsItIvE. For example, be sure to use "S13" instead of "s13".

1 Important Tips when using DrHJ

1. If you use the Interactions Pane in DrHJ, it is very important that you *reset the Interactions Pane after each run of a program.* This can be done by clicking on the "Reset" button on the top, just to the right of "Compile". Failure to do so may lead to unpredictable results for the second run of your program, because it may be impacted by updates to static fields that were made in the first run.

   In particular, if you attempt to run a program with data race detection support enabled multiple times in the same Interactions Pane session, you will see the following message for each run after the first:

   > Error: Race Detection already ran in this instance of the HJ Runtime

   The only fix for this problem is to reset the Interactions Pane before each run.

2. Datarace detection support in DrHJ only works if your *entire program is contained in a single .hj file.* If your program contains multiple .hj files, you will need to use HJ's command-line interface to turn on datarace detection. See optional note below for advanced users

3. When selecting compiler options for DrHJ, make sure to *choose at most one of the following compiler options at a time* — "Show Abstract Execution Metrics" or "Show Race Detection Warnings". Fortunately, the Compiler Options pane in DrHJ's Preferences does not allow both options to be selected simultaneously.

4. If the source location of a compiler error message appears unclear in the "Compiler Output" pane, click on "Console Output" and you may see a precise location in *line:column* format. For example 10:20 refers to column 20 in line 10.

Optional note for advanced users: If you prefer to use a command-line interface instead of DrHJ to compile and run HJ programs, you can down load an HJ installation from the "HJ Download and Setup" page listed above by searching for "Download the zip file containing the HJ package" and then following the subsequent instructions. The command-line interface only works on Unix-based systems (*e.g.,* Linux, Mac OS), and not on Windows. In contrast, DrHJ runs on both Unix-based systems and Windows systems. We will introduce command-line interfaces for HJ to all students later in the semester.

2 Example HJ Program for Data Race Detection

The HJ compiler supports a data race detection option to enable automatic instrumentation of the HJ program to check for data races at runtime. This capability can be enabled by selecting the "Show Race Detection Warnings" option in DrHJ's Compiler Option preferences (see Figure 1), or (if you're using the command line) by typing the following command: *hjc -racedet RacyArraySum1*. After this compilation, the HJ program can be launched as usual. Instead of a normal parallel execution, the instrumented version runs on a single processor core (single worker) 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.

1. Download the `RacyArraySum1.hj` file from the Code Examples column for Lab 3 in the course web page.
2. Compile this HJ program with the appropriate option for enabling datarace detection, as described above.
3. Now, run the program, and observe from the output the exact locations in the program where the data race exists. Insert a finish to fix the data race, and re-run with data race detection enabled to check for additional races until no more races are encountered in your testing. You can consult the ArraySum1.hj example from Lab 2 if you would like to remind yourself where the finish was originally placed.

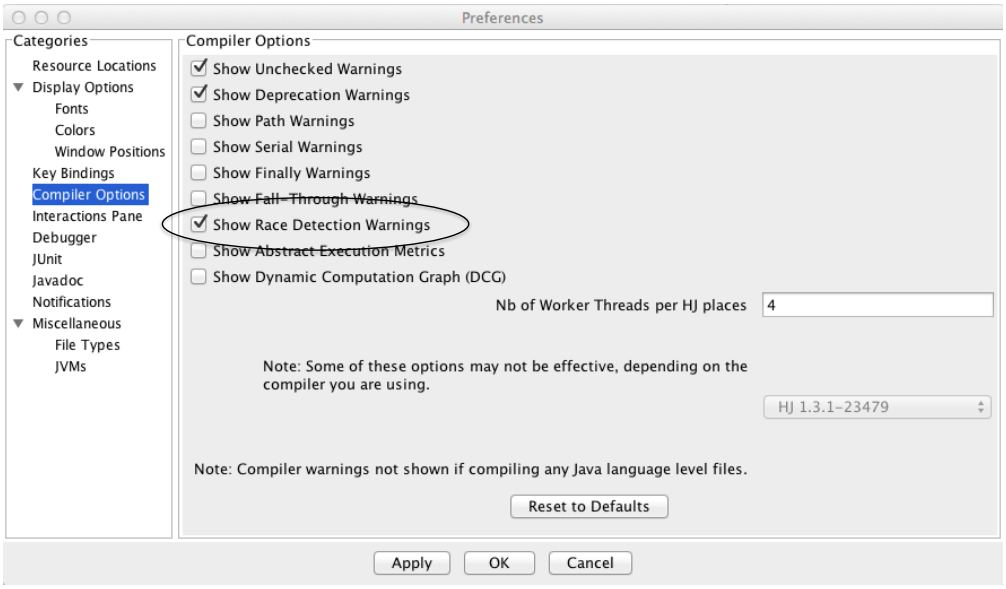

Figure 1: Selection of "Show Abstract Execution Metrics" in DrHJ's Compiler Option preferences

3 Lab 3 Assignment

1. As before, create a text file named `lab_3_written.txt` in the `lab_3` directory. It will also help you to look at the questions in the Lab 3 quiz in Coursera before you start the lab assignment.

2. Check each parallel HJ program that you submitted in Lab 1 and Lab 2 to see if it exhibits a data race. For `ReciprocalArraySum.hj` from Lab 1, we recommend reducing `default_n` from 25 million to 2.5 million (2500000), because the larger size may cause the program to run too slowly with data race detection enabled. For other programs, the default input should be fine. (You can browse `https://svn.rice.edu/r/comp322/turnin/S13/`*your-netid* to locate your past submissions.) Write your answer in `lab_3_written.txt`.

3. Repeat steps 4–9 for each of the remaining files in the Code Examples column for Lab 3 in the course web page.

4. Download the `.hj` file.

5. Compile the program with datarace detection enabled.

6. Run the program, and observe the source code location for conflicting accesses.

7. Edit the program to fix the data race bug (if any) and get the correct output. (You can make a copy of the program and comment out all occurrences of "async" to obtain a correct sequential program, if you'd like to use it as baseline to understand what the correct version of the parallel program is supposed to compute.) You may edit the program by inserting finish statements, moving async statements, or expanding a single-dimensional array to a multi-dimensional array.

8. Repeat steps 5, 6, 7 for the current parallel program until no data race is reported.

9. Create an entry for the `.hj` program in `lab_3_written.txt`. The entry should summarize in 2-3 sentences how much parallelism had to be given up to guarantee data race freedom, and whether the final program has no parallelism *i.e.,* is it equivalent to a sequential program?

4 Turning in your quiz and lab work

As in previous labs, you will need to complete a quiz on Coursera and turn in your work before leaving, as follows:

1. Visit rice.coursera.org, select "Fundamentals of Parallel Programming" course, and take the Lab 3 quiz.

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. Before you leave, create a zip file of your work by changing to the parent directory for `lab_3/` and issuing the following command, "`zip -r lab_3.zip lab_3`".

4. Use the turn-in script to submit the contents of the `lab_3.zip` file as a new `lab_3` directory in your turnin repository as explained in Lab 1. You can always examine the most recent contents of your svn repository by visiting `https://svn.rice.edu/r/comp322/turnin/S13/`*your-netid*.