Lab 4: Futures

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 log in via Shibboleth

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

NOTE: As in previous labs, 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.
2. If the source location of a compiler error message appears unclear in the "Compiler Output" pane, click on "Console Output" and you should 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 Futures: ArraySum2

1. Download the `ArraySum2.hj` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.
2. Compile this HJ program. Click "Compile" in DrHJ, or (if you're not using DrHJ) type the following command on the command line: *hjc ArraySum2.hj*
3. Run the program with an option to generate abstract performance metrics. Select "Show Abstract Execution Metrics" in DrHJ's Compiler Option preferences, or (if you're not using DrHJ) type the following command on the command line: *hj -perf=true ArraySum2*
4. Notice the following statistics printed at the end of program execution for the default array size of 1024:

```
> run ArraySum2
Sum of 1024 elements = 541525 (should be 541525 for n = 1024)

#### START OF ABSTRACT EXECUTION STATISTICS ####
    TOTAL NUMBER OF TASKS = 3072

    WORK = TOTAL NUMBER OF OPS DEFINED BY CALLS TO hj.lang.perf.doWork() = 1023

    CPL = CRITICAL PATH LENGTH OF OPS DEFINED BY CALLS TO hj.lang.perf.doWork() = 10

    IDEAL PARALLELISM = WORK/CPL = 102.3
#### END OF ABSTRACT EXECUTION STATISTICS ####
```

5. You can repeat the run for a different array size by typing "`run ArraySum2` *size*" in DrHJ's Interactions Pane.

6. *As before, create a text file named* `lab_4_written.txt` *in the* `lab_4` *directory.*

7. What WORK, CPL and IDEAL PARALLELISM values do you you see for different array sizes? Enter these values in a file named `lab_2_written.txt` in the `lab_2` directory. for array sizes that range across all powers of 2 up to 128 — 1, 2, 4, 8, 16, 32, 64, 128.

   NOTE: You may encounter the following error message in DrHJ in today's lab, as you may have in Homework 1: "*ERROR: Maximum number of hj threads per place reached (128 threads)*".

   You will learn the detailed reason behind this message in future labs. In the worst case, the default HJ runtime (work-sharing) may need to create additional HJ worker threads each time an end-finish or future-get operations is encountered. There are other HJ runtimes that don't have this limitation, but they currently do not generate abstract performance metrics. The default limit for worker threads is 128 to prevent an HJ job from consuming too many resources on a machine. However, this default can be overridden by using the -MAX_THREADS_PER_PLACE option as follows (by typing this command in the DrHJ Interactions Pane): *run -MAX_THREADS_PER_PLACE=1000 quicksort* .

   One consequence of choosing a large value of MAX_THREADS_PER_PLACE is that your program may run out of memory. In that case, you can increase the memory allocated to the interactions JVM by selecting "JVMs" in the "Miscellaneous" tab in DrHJ preferences, and increasing the amount of memory allocated to the Interactions JVM (not the Main JVM).

3 Array Sum Revisited with Variable Execution Times: ArraySum4

1. Download the `ArraySum4.hj` file from the Code Examples column for Lab 4 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.

2. The main difference compared to `ArraySum2.hj` is that the call to `doWork()` in `ArraySum4.hj` estimates the cost of an add as the number of significant bits in both operands. Thus, the cost depends on the values being added.

3. Again, enter WORK, CPL and IDEAL PARALLELISM values in `lab_4_written.txt` For array sizes that range across all powers of 2 up to 128 — 1, 2, 4, 8, 16, 32, 64, 128. While it is reasonable to see higher WORK and CPL values for ArraySum4 than ArraySum2, comment in `lab_4_written.txt` on whether the IDEAL PARALLELISM is higher or lower for ArraySum4 than ArraySum2 and why that's the case. Also, look at the IDEAL PARALLELISM results that you obtained in Lab 2 for ArraySum3 (see your earlier report in `lab_2/lab_2_written.txt`) and comment in `lab_4_written.txt` on the differences in IDEAL parallelism between ArraySum3 and ArraySum4.

4 Binary Trees using Futures

1. Download the `binarytrees.hj` file from the Code Examples column for Lab 4 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.

2. Compile and run the program and record the WORK and CPL values. (They will be the same since this is a sequential program.)

3. Now modify the program by replacing "`final private TreeNode left;`" by "`final private future<TreeNode> left;`" in line 35 and "`final private TreeNode right;`" by "`final private future<TreeNode> right;`" in line 36. After this modification, you will need to replace references to left and right by using future expressions.

4. After you get your modified program to work, again record the WORK and CPL values, and comment in `lab_4_written.txt` if you've achieved an increase in parallelism, and, if so, why.

5 Turning in your lab work and quiz

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 4 quiz.

2. Check that all the work for today's lab is in the `lab_4` 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_4/` and issuing the following command, "`zip -r lab_4.zip lab_4`".

4. Use the turn-in script to submit the contents of the `lab_4.zip` file as a new `lab_4` 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*.