Lab 9: Java Threads

Instructor: Vivek Sarkar, Due: Friday March 28, 2014

Resource Summary

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

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

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

Important tips and links:

edX site : `https://edge.edx.org/courses/RiceX/COMP322/1T2014R`

Piazza site : `https://piazza.com/rice/spring2014/comp322/home`

Java 8 Download : `https://jdk8.java.net/download.html`

IntelliJ IDEA : `http://www.jetbrains.com/idea/download/`

HJ-lib Jar File : `http://www.cs.rice.edu/~vs3/hjlib/habanero-java-lib.jar`

HJ-lib API Documentation : `https://wiki.rice.edu/confluence/display/PARPROG/API+Documentation`

HelloWorld Project : `https://wiki.rice.edu/confluence/display/PARPROG/Download+and+Set+Up`

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

Linux Tutorial visit `http://www.rcsg.rice.edu/tutorials/`

On SUGAR, JDK8 is already available at `/users/COMP322/jdk1.8.0` and HJ-Lib is already installed at `/users/COMP322/habanero-java-lib.jar`. Run the following command to setup the JDK8 path.

```
source /users/COMP322/hjLibSetup.txt
```

When you log on to Sugar, you will be connected to a *login node* along with many other users. To request a dedicated *compute node*, you should use the following command from a SUGAR login node:

```
qsub -q commons -I -V -l nodes=1:ppn=8,walltime=00:30:00
```

We learned that there may be some issues with SUGAR this week. If you're unable to obtain a compute node, please use your laptop for today's lab instead.

Note that all commands below are CaSe-SeNsItIvE. For example, be sure to use "S14" instead of "s14".

IMPORTANT: please refer to the tutorial on Linux and SUGAR, before staring this lab. Also, if you and others experience long waiting times with the "qsub" command, please ask the TAs to announce to everyone that they should use ppn=4 instead of ppn=8 in their qsub command (to request 4 cores instead of 8 cores).

1 Conversion to Java Threads: N-Queens

1. Download the `nqueens_hj.java` program from the course web site (scroll down to Lab 9). This version uses finish and async constructs along with AtomicInteger calls.

2. Convert it to a pure Java program by using Java threads instead of finish/async, using the concepts introduced in Lectures 24 and 25. (The AtomicInteger calls can stay unchanged.) For simplicity, you can include joins within each call to `nqueens_kernel()`. This is correct, but more restrictive than the finish/async structure for the given code. But it simplifies parallelization when using Java threads.

   If you wish to try and simulate a finish more accurately, you can do so by collecting all thread objects in a `ConcurrentLinkedQueue` data structure (see Lecture 24) and calling `join()` on each of them at the end of the computation.

3. Compile and run the program as follows to solve the N-Queens problem on a 12×12 board (default value).
   ```
   javac nqueens.java
   java nqueens
   ```

4. Compare the execution time of three versions of NQueens:

   (a) `java nqueens 14 5 0`
   This should correspond to the sequential execution of your Java program since the third argument (= 0) is the cutoff value.

   (b) `java nqueens 14`
   This is a parallel Java run with the default cutoff value of 3. Try experimenting with different values for `cutoff_value` if needed.

   (c) `java -cp /users/COMP322/habanero-java-lib.jar:. -Dhj.numWorkers=8 nqueens_hj 14`
   This is a parallel HJlib version run with the default cutoff value of 3. (It is recommended that you use separate directories for compiling the Java and HJ versions so as to avoid any possible interference among classfiles generated for both versions.)
   NOTE: Make sure you have compiled the hj version before running the java command.

2 Conversion to Java threads: Spanning Tree

1. Download the `spanning_tree_atomic_hj.java` solution from the course web site (scroll down to Lab 9). This version uses finish and async constructs along with AtomicReference calls.

2. Convert it to a pure Java program by using Java threads instead of finish/async, using the concepts introduced in Lectures 24 and 25. (The AtomicReference calls can stay unchanged.) As before, you can include joins within each call to `compute()` for simplicity, or you can use a `ConcurrentLinkedQueue` for a more faithful simulation of a finish construct.

3. Compile and run the programs as follows with the default input size.
   ```
   javac spanning_tree_atomic.java
   java spanning_tree_atomic
   ```

4. Compare the execution time of three versions of the spanning tree example. You may choose to add cutoff threshold values for this program as was done for N-Queens, so as to limit the number of Java threads that will be created:

   (a) `java spanning_tree_atomic 50000 1000`
   This is a parallel Java run. If you add support for a `cutoff_value`, you can experiment with different cutoff values.

(b) `java -cp /users/COMP322/habanero-java-lib.jar:. -Dhj.numWorkers=8 spanning_tree_atomic_hj 50000 1000`
This is a parallel HJlib run. If you used a cutoff value for the parallel Java run above, you should also add it for this HJlib version. (It is recommended that you use separate directories for compiling the Java and HJlib versions so as to avoid any possible interference among classfiles generated for both versions.)
NOTE: Make sure you have compiled the hj version before running the java command.

3 Programming Tips and Pitfalls for Java Threads

- Recall that any local variable from an outer scope that is accessed in an anonymous class (e.g., in the run() method) *must be declared final*.
- Remember to call the *start()* method on any thread that you create. Otherwise, the thread's computation does not get executed.
- Since the *join()* method may potentially throw an InterruptedException, you will either need to include each call to *join()* in a *try-catch block*, or add a *throws InterruptedException* clause to the definition of the method that includes the call to *join()*.

4 Turning in your lab work

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

1. Check that all the work for today's lab is in the `lab_9` 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.
2. Use the turn-in script to submit the `lab_9` directory to your turnin directory as explained in the first handout: *turnin comp322-S14:lab_9*. 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_9.zip` *file to comp322-staff@mailman.rice.edu.*