Lab 11: Java Threads

Instructor: Vivek Sarkar, Due: Wednesday April 8, 2015

Resource Summary

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

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

Important tips and links:

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

Piazza site : `https://piazza.com/rice/spring2015/comp322/home`

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

Maven Download : `http://maven.apache.org/download.cgi`

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

HJ-lib Jar File : `http://www.cs.rice.edu/~vs3/hjlib/code/maven-repo/edu/rice/hjlib-cooperative/0.1.5-SNAPSHOT/hjlib-cooperative-0.1.5-SNAPSHOT.jar`

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

HelloWorld Project : `https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124`

1 Lab Goal

In today's lab you will practice using Java Threads.

The Maven project for this lab is located in the following svn repository:

- `https://svn.rice.edu/r/comp322/turnin/S15/`*`NETID`*`/lab_11_threads`

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_11_threads/ lab_11
```

In today's lab, you need to use STIC to submit computation jobs. If you need any guidance on how to submit jobs on STIC, please refer to the appendix at the end of this document or any of the previous labs in which you used STIC.

You also need to configure arguments for your program. If you forget how to do so, please refer to the note on page 1 of lab_9.

2 Conversion to Java Threads: N-Queens

1. The `NQueensThreads.java` program is a sequential solution to the N-Queens problem.

   The `NQueensHjLib.java` program has been implemented for you to solve the N-Queens problem. This version uses finish, async and finish accumulators.

2. Your task is to convert `NQueensHjLib.java` to a pure Java program by using Java threads concepts introduced in Lectures 26 and 27, instead of using the HJ Library. *You should make your changes in the provided* `NQueensThreads.java` *file.*

   For simplicity, you can include joins within each call to `nqueensKernel()`. 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 26) and calling `join()` on each of them at the end of the computation.

3. To test your solution, solve an N-Queens problem on a 12×12 board (default argument value) by compiling and running the program on STIC using the followig command line:

   ```
   mvn clean compile exec:exec -Pnqueens_hj
   mvn clean compile exec:exec -Pnqueens
   ```

   The first command will compile and execute the HjLib version and the second command will compile and execute the thread version.

   You should also run the provided unit tests to make sure that you have a correct solution. *Again, because the provided NQueensThreads.java code is a working sequential solution, the project you check out without any modification will actually pass all tests.*

4. Then on STIC, modify the myjob.slurm template that has been provided to you under:

   `lab_11_threads/src/main/resources`

   to submit lab 11 computation jobs to the computing node.

   Compare the execution time of the following versions of NQueens. (You should modify the arguments in the pom.xml. See page 1 of lab_9 if you forget how to change arguments.)

   (a) `NQueensThreads` with input arguments 14 and 0
   This corresponds to the sequential execution of your Java program since the second argument specifies the cutoff value to be 0.
   Our reference solution has an execution time around 49000ms on average.

   (b) `NQueensThreads` with input arguments just 14
   This is a parallel Java run with the default cutoff value of 3.
   Our reference solution has an execution time around 6500ms on average.
   You can also try experimenting with different values for `cutoff` and observe the changes in execution time.

   (c) `NQueensHjLib` with input argument just 14
   This is a parallel HJLib version run on a 14-by-14 board,with the default cutoff value of 3.
   It should take around 6800ms on average.

3 Conversion to Java threads: Spanning Tree

1. The `SpanningTreeAtomicThreads.java` program is a sequential solution to the problem.

   The `SpanningTreeAtomicHjLib.java` program has been implemented for you to solve the minimum spanning tree problem. This version uses finish and async constructs along with AtomicReference calls.

2. Your task is to convert SpanningTreeAtomicHjLib.java to a pure Java program. You should modify `SpanningTreeAtomicThreads.java` program we have given to you. Use Java threads instead of finish/async. (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 locally as follows with the default input size to test your program.
   ```
   mvn clean compile exec:exec -Pspanning_tree_atomic_hj
   mvn clean compile exec:exec -Pspanning_tree_atomic
   ```

   The first command will compile and execute the HjLib version and the second command will compile and execute the thread version. You should also run the provided unit tests to make sure that you have a correct solution. *Again, because the provided SpanningTreeAtomicThreads.java code is a working sequential solution, the project you check out without any modification will actually pass all tests.*

4. Compare the execution time of following versions of the spanning tree problem. 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) `SpanningTreeAtomicThreads` with input arguments 50000 and 3000
   This is a parallel Java run with 50000 nodes and 3000 neighbors per node. If you add support for a `cutoff_value`, you can experiment with different cutoff values.
   Our reference solution(which uses a cutoff value of 5) has an execution time around 1200ms.

   (b) `SpanningTreeAtomicHjLib` with input arguments 50000 and 3000
   This is a parallel HJlib run. If you choose to use a cutoff value for the parallel Java run above, you should also add cutoff value support for this HJlib version.
   It should take around 1000ms on average.

4 Programming Tips and Pitfalls for Java Threads

- 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()*.

5 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. Be prepared to explain the lab at a high level.

2. Check that all the work for today's lab is in the `lab_11_threads` 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 svn command script to submit the `lab_11_threads` directory to your turnin directory as explained in the first handout. Note that you should *not* turn in a zip file.

Appendix: STIC setup

STIC(Shared Tightly-Integrated Cluster) is designed to run large multi-node jobs over a fast interconnect. The main difference between STIC and CLEAR is that STIC allows you to gain access to compute nodes to obtain reliable performance timings for your programming assignments. On CLEAR, you have no control over who else may be using a compute node at the same time as you.

- Login to STIC.

  ```
  ssh ⟨your-netid⟩@stic.rice.edu
  ⟨your-password⟩
  ```

  Your password should be the same as the one you have used to login CLEAR. Note that this login connects you to a *login* node.

- After you have logged in STIC, run the following command to setup the JDK8 and Maven path.

  ```
  source /home/smi1/dev/hjLibSource.txt
  ```

  Note: You will have to run this command each time you log on STIC. You could choose to add the command in /.bash_profile so that it will run automatically each time you log in.

- Check your installation by running the following commands:

  ```
  which java
  ```

  You should see the following: `/home/smi1/dev/jdk1.8.0_31/bin/java`

  Check java installation:

  ```
  java -version
  ```

  You should see the following:

  ```
  java version "1.8.0_31"
  Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
  Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
  ```

  Check maven installation:

  ```
  mvn --version
  ```

  You should see the following:

  ```
  Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 10:22:22-0500)
  Maven home:  /home/smi1/dev/apache-maven-3.1.1
  Java version:  1.8.0_31, vendor:  Oracle Corporation
  Java home:  /home/smi1/dev/jdk1.8.0_31/jre
  Default locale:  en_US, platform encoding:  UTF-8
  OS name:  "linux", version:  "2.6.18-371.perfctr.el5", arch:  "amd64", family:
  "unix"
  ```

- When you log on to STIC, you will be connected to a *login node* along with many other users. Once you have an executable program, and are ready to run it on the compute nodes, you must create a job script that uses commands to prepare for execution of your program. We have provided a script template in

  ```
  lab_11_threads/src/main/resources/myjob.slurm
  ```

- To submit the job, run the following command in the same directory where you put myjob.slurm(in this case, it was place under `lab5/src/main/resources/myjob.slurm`):

  ```
  sbatch myjob.slurm
  ```

  After you have submitted the job, you should see the following:

  ```
  Submitted batch job [job number]
  ```

- To check the status of a submitted job, use the following command:

  ```
  squeue -u [your-net-id]
  ```

- To cancel a submitted job, use the following command:

  ```
  scancel [job-id]
  ```

  When your job finished running, your should see an output file titled `slurm-[job-id].out` in the same directory where you have submitted the job.

- To transfer a project folder to STIC, you can use one of two methods:

  – Use Subversion: You can commit your local changes to SVN. Then you can checkout or update the project on your STIC account using one of the following:

  ```
  svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_11/
  ```

  or, if you have already checked out the SVN project on your account,

  ```
  svn update
  ```

  – Use SCP: Use the following command on your local machine:

  ```
  scp -r [folder-name] [your-net-id]@stic.rice.edu:[path to the storage location]
  ```

  For example, if I have a folder named "lab11" on my local machine, and I want to store it in "./comp322" on STIC, I would type the following command:

  ```
  scp -r lab11 [net-id]@stic.rice.edu:./comp322
  ```