Lab 9: Isolated Statements, Atomic Variables

Instructor: Vivek Sarkar

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

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

Importants 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 Parallelization using Isolated Construct

A parallelization strategy for the spanning tree algorithm was introduced this week in Lecture 20, along with an introduction to isolated statements. Recall the following constraints on isolated statements — an isolated statement may not contain any HJ statement that can perform a blocking operation e.g., finish, future get(), and phaser next/wait. In addition, a current limitation in the HJ implementation is that it does not support return statements within isolated.

Before you start the lab, recall from Lab 5 that the arguments of your program can be configured in the `pom.xml` file in the profile section. You can configure the command line options in the `pom.xml` file:

```
<configuration>
    <executable>java</executable>
    <arguments>
        <argument>-javaagent:${edu.rice:hjlib-cooperative:jar}</argument>
        <argument>-classpath</argument>
        <classpath/>
        <argument>${SpanningTreeSeq.class}</argument>
        <argument>1000</argument>  <!--# of nodes in graph-->
        <argument>10</argument> <!--# of neighbors-->
    </arguments>
</configuration>
```

Your task is to perform the following for the `SpanningTreeSeq1.java` program provided for the lab.

1. Compile the sequential and run `SpanningTreeSeq1.java` with two arguments, 1000 (number of nodes in graph) and 10 (number of neighbors). We will ask you to using larger input arguments for parallel version of the code later.

2. Parallelize this program by adding async, finish, and isolated constructs as described in Lecture 20. Call the parallelized version `SpanningTreeIsolated.java` (template has been provided)

3. Compile the parallel `SpanningTreeIsolated.java` program and run with 1 and 8 workers with a large problem size using arguments, 100,000 (number of nodes in graph) and 100 (number of neighbors)[1]:

4. Record the best of 5 execution times reported for `SpanningTreeIsolated.java` (1 and 8 workers) in `lab_9_written.txt`. What speedup do you see?

2 Parallelization using Atomic Variables

Atomic Variables were introduced in Lecture 22.

Your task in this section is to modify the `SpanningTreeAtomic.java` program that replaces `isolated` in your `SpanningTreeIsolated.java` version by an equivalent atomic constructs. Compile and execute your program `SpanningTreeAtomic.java` program by repeating the steps from the previous section. Record the resulting performance in `lab_9_written.txt`.

At this time, you are ready to be checked off. Run the unit tests:

```
mvn clean compile test
```

And get checked off.

3 Parallelization using Object-based Isolation (Optional)

Object-based isolation was also introduced in Lecture 20, with the form

```
isolated(obj1, obj2, ..., () -> <body>)
```

where $obj1, obj2, \ldots$ is a list of object references. Your task in this section is create a `SpanningTreeObjectIsolated.java` program that replaces `isolated` in your `SpanningTreeIsolated.java` version by an equivalent object-based isolated construct. Compile and execute your program `SpanningTreeObjectIsolated.java` program by repeating the steps from the previous section. Record the resulting performance in `lab_9_written.txt`.

4 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.

[1] The sequential version will likely encounter a stack overflow for the large problem size, but the parallel version should run to completion successfully on 1 worker.

2. 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.

3. Use the svn command script to submit the `lab_9` 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

  ```
  lab5/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_9/
    ```

    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 "lab9" on my local machine, and I want to store it in "./comp322" on STIC, I would type the following command:

    ```
    scp -r lab9 [net-id]@stic.rice.edu:./comp322
    ```