Lab 7: Isolated Statements, Atomic Variables

Instructor: Vivek Sarkar, Co-Instructor: Shams Imam

Course Wiki: http://comp322.rice.edu

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

Goals for this lab

- Gain experience using the isolated construct introduced in Lecture 20
- Gain experience using atomic variables introduced in Lecture 21
- Compare the performance of parallel implementations of spanning tree construction that use the isolated construct and atomic variables with a sequential implementation

Importants tips and links

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

Piazza site : https://piazza.com/rice/spring2016/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/

HJlib Jar File : https://github.com/habanero-maven/hjlib-maven-repo/raw/mvn-repo/edu/rice/hjlib-cooperative/0.1.8/hjlib-cooperative-0.1.8.jar

HJlib API Documentation : http://pasiphae.cs.rice.edu/

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

1 Spanning Tree Construction

The idea of a spanning tree was introduced/reviewed in Lecture 20. Given a connected and undirected graph `G` consisting of:

1. A set of nodes `N`
2. A set of edges `E` between nodes in `N`
3. An arbitrarily defined root node `R` in `N` for graph `G`

A spanning tree is a set of edges `S` in `E` that touch all nodes in `N` and contains no cycles (i.e. is a tree). To have no cycles, we require that `|S|` be exactly `|N| - 1`.

One way to define a spanning tree over `G` is by defining a "parent" node for each node in `N`, with each node having at most one parent and the root node `R` having no parents.

Constructing a spanning tree is generally phrased as a graph traversal algorithm where we start at the root node `R` and through a depth-first traversal set the parent of each node if it has not already been set. If it has already been set, we know we already explored that node.

Lectures 20 and 21 discussed how this algorithm can be parallelized.

2 Parallelization using Isolated Construct

First, we will parallelize spanning tree construction and use isolated to protect against concurrent accesses.

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.

A reference sequential implementing of spanning tree construction is provided in `SpanningTreeSeq1.java`.

Your first task is to implement parallel spanning tree construction by adding async, finish, and/or isolated constructs as described in Lectures 20 and 21 to the template in `SpanningTreeIsolated.java`. There are guiding `TODOs` in that file.

You can verify the correctness of your isolated version using the test `Lab7CorrectnessTest.testIsolated`. Note that if you run the performance tests at this point you will likely see stack overflow errors.

3 Parallelization using Atomic Variables

Atomic Variables were introduced in Lecture 21.

Your task in this section is to modify the `SpanningTreeAtomic.java` program to run spanning tree construction in parallel and use atomic variables to protect against concurrent accesses, similar to how you added isolated constructs to `SpanningTreeIsolated.java`. There are guiding `TODOs` in that file.

Like the previous section, you can verify the correctness of your atomic version using the test `Lab7CorrectnessTest.testAtomic`.

4 Performance Evaluation on NOTS

The provided `Lab7PerformanceTest.testIsolated` and `Lab7PerformanceTest.testAtomic` tests will measure the performance of your parallel implementations running on 1 worker thread and running on N worker threads, where N is the number of cores in the platform. These measurements will be performed on a graph with 100,000 nodes and 100 neighbors per node. Use the provided `myjob.slurm` file or the autograder to test the performance of your implementation on NOTS. **Record the performance of both your isolated and atomic versions in a lab_7_written.txt file in your lab_7 turnin directory. You can simply copy-paste the test output if you like.**

While you may not see a linear Nx speedup on N cores for either parallel version, you should see some improvement. **If you do not see perfect linear speedup, you should explain why in lab_7_written.txt based on your understanding of isolated and atomics and the way you are using them in your implementation.**

5 Turning in your lab work

For lab_7, 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. In particular, the TAs will be interested in seeing your changes for the isolated and atomic versions as well as the speedup you achieved relative to one thread.
2. Commit your work to your `lab_7` turnin folder. The only changes that must be committed are your modifications to `SpanningTreeIsolated.java` and `SpanningTreeAtomic.java`, as well as your

`lab_7_written.txt` file. Check that all the work for today's lab is in your `lab_7` directory by opening `https://svn.rice.edu/r/comp322/turnin/S16/NETID/lab_7/` in your web browser and checking that your changes have appeared.