Lab 6: Data-Driven Futures and Phasers

Instructor: Vivek Sarkar, Co-Instructor: Shams Imam

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

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

Goals for this lab

- Experimentation with Data-Driven Futures (Abstract Metrics)
- Experimentation with Phasers (Real Performance on NOTS)

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/

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

Lab Projects

Today, we will be using the Cholesky factorization example to learn about using Data Driven Futures. We will compute performance for this part of the lab using abstract metrics. Next, we will revisit the Iterative Averaging example from last lab to implement a point-to-point synchronization variant using phasers. Then, we will compute real performance on the NOTS cluster at Rice.

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

- `https://svn.rice.edu/r/comp322/turnin/S16/`*`NETID`*`/lab_6`

or can be downloaded from the COMP 322 website. Please pull this project down, import it into IntelliJ, and verify you can build it. Feel free to use whatever methods you are most comfortable with to achieve this (e.g. command-line SVN vs. IntelliJ SVN, automatic Maven-based JAR configuration vs. manual JAR imports, etc). If you need them, instructions are available in the HW2 handout. As always, be sure that the HJlib `-javaagent` command line option is added to any run configurations you use in IntelliJ.

1 Reminder: NOTS setup

NOTS (Night Owls Time-Sharing Service) is a Rice compute cluster designed to run large multi-node jobs over a fast interconnect. The main difference between using NOTS and using your laptop is that NOTS allows you to gain access to dedicated compute nodes to obtain reliable performance timings for your programming assignments. On your laptop, you have less control over when other processes or your power manager might start stealing cores or memory from your running parallel program.

Prior to lab, you should have completed the setup instructions from https://piazza.com/class/iirz0u74egl2q9?cid=151. If you have not, please do so now. These instructions ensure that we have 1) an SSH client that allows us to remotely access a terminal on NOTS, and 2) some method of uploading and downloading files to NOTS (either through SCP or SFTP). Here is a summary of the steps (please see the detailed instructions from previous lab handout):

- Start by logging in to NOTS ().
- After you have logged in to NOTS, setup the JDK8 and Maven path.
- Check your installation by running some commands.
- To obtain performance results create a job script that configures and runs your program on a dedicated compute node. We have provided a script template in the Lab 5 template code at:

  `lab_6/src/main/resources/myjob.slurm`

  You only need to change the line marked "TODO". The change on line 8 of that file indicates an e-mail address to send job status notifications to.
- Transfer your `lab_6` folder from your local machine up to NOTS so that we can submit the lab_6 code to the cluster for testing.
- Submit jobs to the NOTS cluster using the `sbatch` command.

2 Parallelization of Cholesky (using Data-Driven Tasks)

In linear algebra, the Cholesky factorization is a decomposition of a positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. This decomposition is useful for efficient numerical solutions, it is mainly used for the numerical solution of linear equations $Ax = b$.

You are provided the sequential version of Cholesky Factorization in the `CholeskyFactorization.java` and `CholeskyFactorizationSequential.java` files. Your assignment is to parallelize the computation using data-driven tasks, and to evaluate the parallelization using abstract metrics. The fact that the sequential version uses a generic `dataStore` container can be leveraged to simplify this conversion.

Your assignment is to create a parallel version of `CholeskyFactorizationSequential.runComputation()` that implement the Cholesky Factorization algorithm but using Data-driven tasks:

1. Write a parallel version in the `runComputation()` method of the `CholeskyFactorizationParallel.java` file using data-driven tasks with calls to `asyncAwait()`. The provided file has helpful hints to guide you in this process. `CholeskyFactorizationParallel.java` is set up with zero overhead for async tasks, as in Homework 1 (but not Homework 2).
2. Run the unit tests to verify whether your parallel computation achieves the required abstract metrics.
3. Record in `lab_6_written.txt` the best total work and the speedup w.r.t to the sequential version.

3 One-Dimensional Iterative Averaging (with Phasers)

The code provided in `OneDimAveraging.java` performs the iterative averaging computation discussed in the lectures (see Lecture 11). This code performs a sequential version of the computation in method `runSequential()`. Iterative averaging is performed on a one-dimensional array of size `(n+2)` with elements 0 and `n+1` initialized to 0 and 1 respectively. The final value expected for each element i at convergence is $i/(n+1)$. However, we limit `iterations` to a constant value to prevent long execution times so you may not reach convergence.

1. *Your assignment is to create a parallel version of `OneDimAveraging.runSequential()` that implements the same one-dimensional iterative averaging algorithm but using phasers:* `runChunkedPhaser`: Use the algorithm presented in Lecture 15 and Lecture 16 to compute `myNew` in parallel. In particular, pay attention to the phaser registration modes for your tasks. Also, you will need to use chunking to obtain good performance in your programs.
2. You can implement the parallel version of the one-dimensional iterative averaging algorithm locally on your laptop, but to complete the lab you must also upload your project to NOTS and run it in a compute job to evaluate the performance of the different parallelization approaches. On NOTS, you should achieve ∼2x speedup relative to the provided sequential version. Use the performance numbers from the previous lab to compare your performance with the phaser-based version and note them down in `lab_6_written.txt`.

4 Autograder

This lab is also supported on the autograder. While we haven't used the autograder for performance testing in the past, it does support submitting batch jobs to the NOTS cluster for you. While the upload process remains the same, you will see a new `TESTING PERFORMANCE` status if you submit to the `COMP322-S16-Lab6` module. On completion, the results of the jobs on the cluster will be displayed on the run info page under the heading `Performance.8`.

5 Turning in your lab work

For lab_6, 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 code for the phaser-based version and the difference between using different versions of phaser registration.
2. Commit your work to your `lab_6` turnin folder. The only changes that must be committed are your modifications to `CholeskyFactorization.java` and `OneDimAveraging.java`. Check that all the work for today's lab is in your `lab_6` directory by opening `https://svn.rice.edu/r/comp322/turnin/S16/NETID/lab_6/` in your web browser and checking that your changes have appeared.