Lab 1: Async-Finish Parallel Programming

Instructor: Vivek Sarkar. Co-Instructor: Shams Imam

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

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

Goals for this lab

- Subversion integration
- Introduction to HJlib installation and setup
- Three HJlib APIs: `launchHabaneroApp`, `async`, and `finish`.
- Autograder submission.

Importants tips and links

NOTE: The instructions below are written for Mac OS and Linux computers, but should be easily adaptable to Windows with minor changes e.g., you may need to use \ instead of / in some commands.

Note that all commands below are CaSe-SeNsItIvE. For example, be sure to use "S16" instead of "s16".

1 Subversion Setup

Subversion is a software versioning and revision control system. A revision control system is a system that tracks incremental revisions of files. It manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data or examine the history of how your data changed. For our purposes, we are mostly interested in using the command-line client program `svn` (svn is an abbreviation for Subversion). You will need to ensure that the `svn` command has been installed on your system. Instructions for installing the command are available at Apache Subversion Binary Packages page. Once installed, you can verify that the command works by running the following command:

```
 $ svn --version
svn, version 1.8.11 (r1643975)
   compiled Jan  5 2015, 13:33:40 on x86_64-apple-darwin13.4.0
   ...
```

Subversion implements the concept of a version control repository - the core storage mechanism for versioned data. Each of you has a private repository for COMP 322 allocated in a "cloud" hosted by Rice's subversion (svn) server, svn.rice.edu. You can always examine the most recent contents of your svn repository by visiting `https://svn.rice.edu/r/comp322/turnin/S16/`*your-netid. It is possible that your svn account is not properly set up as yet. If you are unable to access the above URL, please send email to helpdesk@rice.edu cc'ing comp322-staff@mailman.rice.edu and requesting that they fix your access. After that, you can ignore this section for now (till you get access) and move on to the next section.*

The svn repository is empty to begin with, but will be populated with folders for homeworks and labs. We have a strict naming convention for these folders — "`hw_1`", "`hw_2`'', ... for homeworks and "`lab_1`", "`lab_2`", ... for labs. You will need to *commit* your local changes to the svn repository. There are a few ways in which you can submit files to your subversion repository for lab and homework submissions:

1. You can integrate subversion with your individual IntelliJ projects; after the integration you can use IntelliJ's GUI to publish your data to svn. Further details are available in online tutorials at `https://www.jetbrains.com/idea/help/version-control-with-intellij-idea.html`. In particular, pay attention to the "Enabling Version Control" and "Common Version Control Procedures" articles.
2. Eclipse users can consider using Subclipse plugin.
3. If you are familiar with subversion and have your own svn client on your local machine, you are welcome to use that instead to commit your files.
4. Another option is to use the command line for svn, i.e. the `svn commit` command. Please refer to the manual for the svn commit command.

NOTE: If you have problems with any homework or lab submission during the semester, just email your submission zip file to comp322-staff@mailman.rice.edu before the deadline.

2 Lab 1 Exercises

The files for this lab should already be available in your subversion server for this course. Please visit the url at `https://svn.rice.edu/r/comp322/turnin/S16/`*your-netid*`/lab_1` to ensure that the files are available. The directory structure for the lab should look like something as follows:

```
README
pom.xml
src
    main
     java
         edu
             rice
                 comp322
                     HelloWorldError.java
                     ReciprocalArraySum.java
    test
        java
            edu
                rice
                    comp322
                        Lab1CorrectnessTest.java
```

The README file contains some helpful tips on commands you might need to run. For this lab, you will be required to only edit the `HelloWorldError.java` and `ReciprocalArraySum.java` files.

You will need to download the project from the subversion server into your machine; set up the project on IntelliJ; and submit your changes back the Rice subversion server. Note that you will need to have Java 8, Maven, and IntelliJ installed for this step. You can download the `lab_1` project on your machine using the following methods:

1. Download the project using the IntelliJ support for Subversion (Instructions with Images).
2. Download the project using the command line (Demo Video).
3. If you do not have subversion set up on your machine, you can download the lab_1.zip file and manually set up the project on IntelliJ. Then you can use the `svn commit` command to submit your changes.

2.1 HelloWorld program

The first exercise is to familiarize yourself with the kind of code you will see and be expected to write in your assignments. The `HelloWorldError.java` program does not have any interesting parallelism, but introduces you to the starter set for HJlib, which consists of three method calls[1]:

- `launchHabaneroApp()` Launches the fragment of code to be run by the Habanero runtime. All your code that uses any of the Habanero constructs must be (transitively) nested inside this method call. For example,

  ```
  launchHabaneroApp(() -> {S1; ...});
  ```

  executes S1, ..., within an implicit `finish`. You are welcome to add `finish` statements explicitly in your code in statements S1, ...

- `async` contains the API for executing a Java 8 lambda asynchronously. For example,

  ```
  async(() -> {S1; ...});
  ```

  spawns a new child task to execute statements S1, ... asynchronously.

- `finish` contains the API for executing a Java 8 lambda in a finish scope. For example,

  ```
  finish(() -> {S1; ...});
  ```

  executes statements S1, ..., but waits until all (transitively) spawned `async`s in the statements' scope have terminated.

For details on all of your favorite HJlib APIs, we recommend bookmarking the all-new and beautiful HJlib documentation website, hosted at `http://pasiphae.cs.rice.edu`.

Start by trying to compile the project by running `mvn clean compile` from the top-level directory, or using IntelliJ to build it. You should receive an error similar to the following:

```
ReciprocalArraySum.java:44:44: error: Parameter name 'Args' must match pattern '^[a-z][a-zA-Z0-9]*$'.
```

This error is being reported by checkstyle. Like in COMP 215, in COMP 322 we will use checkstyle to encourage clean coding habits. This is good practice for any of you planning for future careers in industry. When working at a company, compliance to style guides will not be a suggestion but rather a requirement for upstreaming any of your code.

Your first step is to fix this error and recompile, ensuring that no checkstyle errors are reported for your project. Note: Your lab and assignment submissions will be required to comply with our provided Checkstyle rules.

Once your checkstyle report is clean, attempting to run the `HelloWorldError.java` program should result in an error that reads like:

```
cannot find symbol variable ss
```

Your first task is to check your setup by fixing this error. You can replace "ss" by "s" and run the program again, verifying that it completes.

[1] Note that these and other HJlib APIs make extensive use of Java 8 lambda expressions.

2.2 ReciprocalArraySum Program

We will now work with the simple two-way parallel array sum program introduced in the Demonstration Video for Topic 1.1. You will edit the `ReciprocalArraySum.java` program for this exercise.

- The goal of this exercise is to create an array of `N` random double's, and compute the sum of their reciprocals in three ways:
  1. Sequentially in method seqArraySum()
  2. In parallel using two tasks in (the currently sequential) method parArraySum() with two loops in lines 151 to 153 for lower and upper halves of the array. The profitability of the parallelism depends on the size of the array and the overhead of async creation. (The overheads have a larger impact for smaller array sizes.) Your assignment is to use two-way parallelism in method parArraySum() to obtain a smaller execution time than seqArraySum(). You are allowed to use the `async` and `finish` constructs in parArraySum().
  3. (Optional) In parallel, using more than two tasks, to see if you can do better than the times obtained with two tasks. Note that the use of an excessive number of tasks can lead to slowdowns due to the overhead of async creation. Further, for this example, it will not help to create more tasks than the number of hardware threads available on your machine. You can check this number by calling the following JDK method, `Runtime.getRuntime().availableProcessors()`.
- Compile and run the program in IntelliJ to ensure that the program runs correctly without your changes. Follow the instructions for "Step 4: Your first project" in https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124.

  If you are not using IntelliJ, you can do this by running the `mvn clean compile exec:exec -Preciprocal` command as specified in the README file.
- Edit the current version to add two-way parallelism to method parArraySum(). Run the unit test either using IntelliJ or via the Maven command line (`mvn clean compile test`) to ensure your solution works as expected. This unit test is designed to fail if the speedup (ratio of sequential to parallel time) is less than 1.5×. The unit test runs the computation for 10 iterations to minimize the impact of dynamic just-in-time compilation, and uses the timings from the 10th iteration to check the speedup. The unit test also checks that the sequential and parallel versions return the same output.
- Experiment with different sizes (specified as an integer `N`) *e.g.,* 10^6, 10^7, and 10^8. You can choose to change the value of `DEFAULT_N` or pass command-line arguments in IntelliJ. NOTE: You may get an `OutOfMemoryError` when experimenting with large values of `N`. If you get an `OutOfMemoryError`, try smaller values of `N` — you will learn how to address that issue in a future lab.
- What speedup (ratio of sequential to parallel time) do you see for different values of `N`? Enter the execution times and speedups in a file named `lab_1_written.txt` in the `lab_1` directory, for different values of `N`.

2.3 Submitting to the Habanero AutoGrader

In COMP 322 we will use an autograding system to test and evaluate lab and homework assignments. Today, you will do a test submission to the Habanero AutoGrader to become familiar with the submission process. The goal is not to produce a submission that passes all tests (in fact, the tests as written will intrinsically not pass when run through the Autograder) but to simply complete a submission.

You should have received an e-mail prior to the start of lab with your login credentials for the autograder. In a web browser, navigate to http://ananke.cs.rice.edu. We recommend using either Firefox or Chrome, IE may cause issues. Enter the provided login credentials and you should be greeted with an empty Overview page.

You will be submitting the completed Maven project that you used to complete the previous sections. The submission process consists of the following steps:

1. Create a zip file containing your full Maven project, including the src/ directory, README, and pom.xml.
2. In your browser, select "COMP322-S16-Lab1" in the "Select Assignment" dropdown.
3. From the file dialog created by clicking the "Browse..." button, navigate to your .zip file and select it for upload to the autograder.
4. Finally, click the "Run" button and wait for your submission to be uploaded. Your run may take some time to complete, particularly if many students submit to the autograder at the same time. Be patient, but please alert a TA if you have been waiting for more than a few minutes or receive any error messages you do not understand.

Once this process is completed, you should see a new entry in the list of "Past Runs" on the Overview page. In the leftmost column is a unique run ID. The center column lists the assignment for the run. The rightmost column provides a status for the run. For today's lab, that status can be `TESTING CORRECTNESS`, `FAILED`, or `FINISHED`.

`TESTING CORRECTNESS` implies that your run is currently being processed. The autograder will test your submission with unit tests, run your submission through a style checker, and use static tools to check for bugs in your submission. For labs and homework, only the correctness, style, and performance of your submission may be counted towards your grade. The various static code checking tools the autograder supplies are purely for your benefit to help you improve your submission.

`FAILED` implies that a failure has occurred while processing your submission. This may indicate a compilation error, an unexpected exit condition from a third-party tool, or an internal error in the autograder. The autograder will not mark your submission as `FAILED` simply because a unit test failed, `FAILED` indicates that a more critical error prevented complete processing of your submission.

`FINISHED` indicates that your run has completed execution and the results can be viewed. To view the results, click on the hyperlink for that run in the "Run ID" column. Among other things, the opened page will list:

1. Your score.
2. The output of the checkstyle tool when run on your program (`checkstyle.txt`).
3. The output of compiling your program (`compile.txt`).
4. The output of your unit tests (`correctness.txt`).
5. The output of the FindBugs static checker (`findbugs.txt`).

For this lab, the expected score is a 28.57%, or 4/14. Even with your fixes to parArraySum, the Lab1CorrectnessTest.testSimple unit test should still fail when run through the autograder and the output from it should report that your parallelized implementation does not run faster than the sequential version. The reasons why are beyond the scope of this lab, you will come to understand why later in the course.

3 Demonstrating and submitting in your lab work

For each lab, you will need to demonstrate and submit 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). They will want to see your files submitted to Subversion in your web browser, the passing unit test on your laptop, and the failed unit test in the autograder UI.

2. Check that all the work for today's lab is in the `lab_1` directory. If not, make a copy of any missing files/folders there.

3. Submit all your changes in the `lab_1` directory using subversion as explained in Section 1. Remember to explicitly add any new files (e.g. your report or any new Java files) you created into the subversion repository.