Lab 1: Infrastructure Setup, Async-Finish Parallel Programming

Instructor: Vivek Sarkar

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

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

Goals for this lab

- Java 8 installation
- Maven installation
- IntelliJ installation
- Subversion integration
- Introduction to HJlib installation and setup
- Three HJlib APIs: `launchHabaneroApp`, `async`, and `finish`.

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 "S15" instead of "s15".

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 : `http://www.cs.rice.edu/~vs3/hjlib/code/maven-repo/habanero-java-lib/hjlib-cooperative/0.1.4-SNAPSHOT/hjlib-cooperative-0.1.4-SNAPSHOT.jar`

HJlib API Documentation : `https://wiki.rice.edu/confluence/display/PARPROG/API+Documentation`

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

1 edX Setup

We will use COMP 322's edX site, `https://edge.edx.org/courses/RiceX/COMP322/1T2014R`, for hosting videos and quizzes. Please only register for COMP 322 on edX with your email address of the form, *your-netid*@rice.edu.

If you can access the above edX site, you are done and you can move on to the next section. If not, please send email to comp322-staff@mailman.rice.edu with a request to add your email address to the edX enrollment list for this class.

2 Piazza Setup

We will use COMP 322's Piazza site, `https://piazza.com/rice/spring2015/comp322/home`, for all discussions and Q&A. Please only register on this site with your email address of the form, *your-netid*@rice.edu.

If you can access the Q&A tab in this Piazza site, you are done and you can move on to the next section. If not, please send email to comp322-staff@mailman.rice.edu with a request to add your email address to the Piazza enrollment list for this class.

3 Java 8 Setup

You may already have Java 8 installed on your computer from COMP 215. Note that we will not be using StratoCode in COMP 322 this semester. To check whether you have Java 8 installed on your machine, go to the command line and type the following `java -version`. If you see something as follows:

```
 $ java -version
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

where the Java version shows 1.8.*, you already have Java 8 installed on your machine. *If you do have Java 8 already installed on your machine, please skip to section 4.*

If you do not already have it installed, you will need a Java 8 installation on your machine and have your `JAVA_HOME` and `PATH` point to the new installation. Java 8 can be downloaded and installed from the Oracle website.

For example, I have the following on my Mac machine's `.bash_profile`:

```
  JAVA8_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home
  export JAVA_HOME=${JAVA8_HOME}
  export PATH=$JAVA_HOME/bin:$PATH
```

On Windows the environment variables have to be set up differently, refer to this stackoverflow question to see how it can be done.

4 Maven Setup

Maven is a build automation tool used primarily for Java projects. Projects using Maven or other build systems are easiest to use as they simplify compiling, building, and testing the project. In addition, major IDEs like IntelliJ and Eclipse have excellent support via maven plugins which simplifies the development process. Dependency management is one of the areas where Maven excels; for our purposes, this means that we will save a lot of effort in configuring the projects to set up dependencies on HJlib, JUnit, and other jars. Hence, for the labs and assignments in COMP 322, we will distribute maven project templates to be used by students to complete their work.

Maven is a Java tool, so you must have Java installed in order to proceed. Next, follow the instructions on the maven site to install maven. Please remember to install a version that is 3.1.1 or later. Detailed instructions are also available on the COMP 322 wiki at `https://wiki.rice.edu/confluence/display/PARPROG/Using+Maven+for+HJlib+projects`. During installation have your `M2_HOME`, `MAVEN_HOME` and `PATH` point to the new installation. For example, I have the following on my Mac machine's `.bash_profile`:

```
export M2_HOME=/Users/shamsimam/dev/apache-maven-3.1.1
export MAVEN_HOME=/Users/shamsimam/dev/apache-maven-3.1.1
export PATH=$PATH:$MAVEN_HOME/bin
```

Once installed, open a new command prompt and run `mvn --version` to verify that it is correctly installed. If you see something as follows:

```
 $ mvn --version
Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 10:22:22-0500)
Maven home: /Users/shamsimam/dev/apache-maven-3.1.1
Java version: 1.8.0_25, vendor: Oracle Corporation
...
```

Maven has been successfully installed on your machine.

5 Use of JUnit in COMP 322

JUnit is a unit testing framework for the Java programming language. As in COMP 215, we will use JUnit for all labs and programming assignments. You will be provided a JUnit test for this lab. Note that, in Maven projects, there are separate directories for the source code and unit tests. Since Maven will handle our dependencies, it will automatically download the necessary JUnit jars. We will be using version 4.7 of JUnit, this can be determined by checking the version in the `pom.xml` file of each individual Maven project.

6 Habanero Java library (HJlib) Installation and Setup

HJlib is a pure Java 8 library implementation of the HJ constructs you have been learning in class. It relies on the use of Java 8 lambdas to simplify writing parallel programs. This library is responsible for scheduling and synchronizing an unbounded number of async tasks on a fixed number of processors ("worker threads"). In COMP 322, we will use two of the scheduling runtime systems in HJlib:

1. *Cooperative Runtime:* The cooperative runtime creates continuations for blocking constructs in HJlib (such as the end of a finish scope), and uses a fixed number of worker threads to switch between async tasks and continuations that are ready to execute. The advantage of the cooperative runtime is that it can support large numbers of async tasks with blocking operations without running out of memory. The disadvantage is that it can cause all methods with blocking constructs (those with a "throws SuspendableException" clause) to run slower than in the normal sequential case. The cooperative runtime is the default for this lab.

2. *Blocking Runtime:* The blocking runtime avoids the overhead of creating continuations, by instead creating a new worker thread each time a blocking operation is performed. The advantage of the blocking runtime is that it incurs less overhead than the cooperative runtime, for programs with small numbers of blocking operations. The disadvantage is that it can easily run out of memory when large numbers of blocking operations are performed (due to the creation of large numbers of worker threads). You will learn options to enable the Blocking Runtime in future labs.

You can follow the download and installation instructions (Step 2 onwards) from the wiki: `https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124`. In summary, the instructions ask you to:

- Install an IDE like IntelliJ IDEA which supports Java 8 lambda syntax. An IDE is not strictly required since Java programs can also be written using a text editor and then compiled using the command-line. However, we strongly recommend using an IDE as it simplifies writing, compiling, running, and debugging your programs. A good IDE gives you error warnings, code completion and navigation, refactoring, syntax highlights, etc.
- Download the HJlib jar file from the url provided in the wiki and save it in a directory of your choice.
- Setup your first HJlib project and get the `HelloWorld` program running on IntelliJ.

7 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/S15/`*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.

8 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/S15/`*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
                        Lab1Test.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.

8.1 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, ...

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

- `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 `asyncs` in the statements' scope have terminated.

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.

8.2 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().
  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're not use 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`.

9 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).
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 7. Remember to explicitly add any new files (e.g. your report or any new Java files) you created into the subversion repository.