Lab 5: Loop Chunking and Barrier Synchronization

Instructor: Vivek Sarkar

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

Staff Email : comp322-staff@rice.edu

Goals for this lab

- STIC setup
- Experimentation with Loop Chunking
- Experimentation with Barriers

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/habanero-java-lib/hjlib-cooperative/0.1.4-SNAPSHOT/hjlib-cooperative-0.1.4-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`

Lab Projects

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

- `https://svn.rice.edu/r/comp322/turnin/S15/`*`NETID`*`/lab_5_onedimavg/`

Please use the subversion command-line client to checkout the project into appropriate directories locally. For example, you can use the following commands from a shell:

```
$ cd ~/comp322
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_5_onedimavg/ lab\_5
```

1 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
  ```

You only need to change the lines marked "TODO". For example, on line 3, change the TODO to the number of nodes you want to run your parallel program on.

- 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_5_ddfs_and_futures/
    ```

    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 "lab5" on my local machine, and I want to store it in "./comp322" on STIC, I would type the following command:

    ```
    scp -r lab5 [net-id]@stic.rice.edu:./comp322
    ```

2 One-Dimensional Iterative Averaging

1. The code in `OneDimAveraging.java` performs the iterative averaging computation discussed in the lectures (see Lecture 12). This code performs a sequential version of the computation in method `runSequential()`.

2. *Your assignment is to create four parallel versions of runSequential() that implement the four versions (with or without chunking/barriers) discussed in the worksheet for Lecture 12.*

3. The input arguments for the main method in this program are as follows:

   (a) `tasks` = number of chunks to be used for chunked parallelism. The default value for `tasks` is 8.

   (b) `n` = problem size. 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 is $i/(n+1)$. The default value for n is 100,000.

   (c) `iterations` = number of iterations needed for convergence. The default value is 5000. This default was set for expediency. For this synthetic problem, you typically many more iterations to guarantee convergence.

   (d) `rounds` = number of repetitions for the entire computation. As discussed earlier, these repetitions are needed for timing accuracy. The default value is 5. For 5 repetitions, a reasonable approach is to just report the minimum time observed.

   The arguments 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:${edu.rice:hjlib-cooperative:jar}</argument>
        <argument>-classpath</argument>
        <classpath/>
        <argument>${OneDimAvg.class}</argument>
        <argument>8</argument>  <!--# tasks for parallel run-->
        <argument>100000</argument> <!--Array size-->
        <argument>5000</argument> <!--# iterations-->
        <argument>4</argument> <!--# rounds of execution timings-->
    </arguments>
</configuration>
```

4. You should run your program on STIC, to evaluate the parallelization. As before, you can compile the program and run the unit tests as follows:

```
mvn clean compile test
```

   To run the program on a *login node*, use the following command:

```
mvn clean compile exec:exec -POneDimAvg
```

   To run the program on a *compute node*, use the following command:

```
sbatch myjob.slurm
```

5. Record in `lab_5_written.txt` the best barrier and phaser-parallel times observed for the default inputs (using 8 cores), and then compute their ratio as the speedup. Compare the results that you obtained for `runParallelForSeqForall()`, `runParallelForSeqForallChunked()`, `runParallelForAllForSeq()`, and `runParallelForAllChunkedForSeq()`.

3 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.
2. Check that all the work for today's lab is in the `lab_5_onedimavg` 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 turn-in script to submit the `lab_5_onedimavg` directory to your turnin directory as explained in the first handout: *turnin comp322-S15:lab_5_onedimavg*. Note that you should *not* turn in a zip file.

   NOTE: Turnin should work for everyone now. If the turnin command does not work for you, please talk to a TA. As a last resort, you can create and email a `lab_5_onedimavg.zip` *file to comp322-staff@mailman.rice.edu.*

Figure 1: myjob.slurm

```
#!/bin/bash

#SBATCH --job-name=lab5
#SBATCH --nodes=2 # TODO: Number of nodes you want your program to run on
#SBATCH --ntasks-per-node=1
#SBATCH --mem=1000m
#SBATCH --time=00:30:00
#SBATCH --mail-user=NETID@rice.edu # TODO: Input your netid
#SBATCH --mail-type=ALL
#SBATCH --export=ALL
#SBATCH --partition=classroom

echo "My job ran on:"
pwd

echo SLURM_NODELIST=$SLURM_NODELIST
echo USER=$USER

cat $SLURM_NODELIST
if [[ -d /home/$USER && -w /home/$USER ]]
then
    cd /home/$USER/lab_5_onedimavg # TODO: Path to your lab 5 folder

    source  /home/smi1/dev/hjLibSource.txt
    java -version
    mvn --version

    mvn clean compile exec:exec -POneDimAvg
fi
```