Lab 4: Real Performance from Loop-Level Parallelism

Instructor: Vivek Sarkar

Resource Summary

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

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

Clear Login: ssh *your-netid*@ssh.clear.rice.edu and then login with your password

Important tips and links:

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

Piazza site : `https://piazza.com/rice/spring2014/comp322/home`

Java 8 Download : `https://jdk8.java.net/download.html`

IntelliJ IDEA : `http://www.jetbrains.com/idea/download/`

HJ-lib Jar File : `http://www.cs.rice.edu/~vs3/hjlib/habanero-java-lib.jar`

HJ-lib API Documentation : `https://wiki.rice.edu/confluence/display/PARPROG/API+Documentation`

HelloWorld Project : `https://wiki.rice.edu/confluence/display/PARPROG/Download+and+Set+Up`

Sugar Login: ssh *your-netid*@sugar.rice.edu and then login with your password

Linux Tutorial visit `http://www.rcsg.rice.edu/tutorials/`

As indicated earlier in a lecture, adding this call in your program before the call to `initializeHabanero()` will increase the limit on blocked threads:
System.setProperty(HjSystemProperty.maxThreads.propertyKey(), "200");

Note that all commands below are CaSe-SeNsItIvE. For example, be sure to use "S14" instead of "s14".

IMPORTANT: please refer to the tutorial on Linux and SUGAR, before staring this lab. Also, if you and others experience long waiting times with the "qsub" command, please ask the TAs to announce to everyone that they should use ppn=4 instead of ppn=8 in their qsub command (to request 4 cores instead of 8 cores).

1 One-time Setup on SUGAR

You should have an account on SUGAR (`http://www.rcsg.rice.edu/sugar`), which is a cluster of Intel Xeon machines, similar to CLEAR. The main difference is that SUGAR allows you to gain dedicated access to compute nodes (see qsub command below) 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 SUGAR.

  ```
  ssh ⟨your-netid⟩@sugar.rcsg.rice.edu
  ⟨your-password⟩
  ```

You should have received an email with the default password convention for Sugar. Use that password when you login the first time. You will be asked to set a new password immediately. Note that this login connects you to a *login* node.

- On SUGAR, JDK8 is already available at `/users/COMP322/jdk1.8.0` and HJ-Lib is already installed at `/users/COMP322/habanero-java-lib.jar`. Run the following command to setup the JDK8 path.

  ```
  source /users/COMP322/hjLibSetup.txt
  ```

- Check your installation by running the following commands:

  ```
  which java
  ```

  You should see the following: `/users/COMP322/jdk1.8.0/bin/java`

  ```
  java -version
  ```

  You should see the following:

  ```
  java version ‘‘1.8.0’’
  Java(TM) SE Runtime Environment (build 1.8.0-b128)
  Java HotSpot(TM) 64-Bit Server VM (build 25.0-b69, mixed mode)
  ```

- When you log on to Sugar, you will be connected to a *login node* along with many other users. To request a dedicated *compute node*, you should use the following command from a SUGAR login node:

  ```
  qsub -q commons -I -V -l nodes=1:ppn=8,walltime=00:30:00
  ```

  When successful, it will give you a command shell on a dedicated 8-core compute node for your use for 30 minutes at a time. Your home directory is the same on both the login and compute nodes.

  For now, just type "*exit*" immediately after you obtain the compute node. We will repeat this command later.

  We only have 12 nodes available for dedicated use for COMP 322. When this limit is exceeded, your request for compute nodes will be pooled with other requests at Rice, which may result in delays. Therefore, it is *very* important that you restrict your use of compute nodes to performance timings. General edit, compile, and debug of HJ-Lib programs can be done on any other computer, or on the SUGAR login nodes. However, please make sure that you don't run any long jobs (> 1 minute) on a SUGAR login node.

2 Experimenting with the asyncSeq clause

The `asyncSeq()` clause is used in Nqueens.java to limit parallelism when *depth* >= *cutoff_value*. The default cutoff is 4, but it can be changed via command line arguments. First you need to compile the Nqueens.java program as follows:

```
javac -cp /users/COMP322/habanero-java-lib.jar Nqueens.java
```

For example, to run the program with 8 workers and a cutoff value to 4, type:

```
java -cp /users/COMP322/habanero-java-lib.jar:.  -Dhj.numWorkers=8 Nqueens 12 3 4
```

since the default arguments were "13 4 6".

Rerun the program with 8 workers with different cutoff values in the range $0 \ldots 12$ and see which one yields the best time for 8 workers.

As in past labs, create a text file named `lab_4_written.txt` *in the* `lab_4` *directory, and enter your timings and observations there.*

3 One-Dimensional Iterative Averaging Example

1. Download the `OneDimAveraging.java` file from the Code Examples column for Lab 4 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.

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

3. *Your first assignment is to create a parallel version of the sequential version in* `runForkJoin()`. *This method has already been provided for you, but its code is the same as* `runSequential()` *which needs to be edited. The API documentation available on the website explains how to use* `forall` *the construct. Remember that* `forall (...) ...` *is the same as* `finish forasync (...) ...`*, you may also choose to use the* `forasync` *construct. Hint: you should have the following structure in your parallel version:* `for-forall–for`.

4. *Your second assignment is to create a parallel chunked version of the sequential version in* `runChunkedForkJoin()`. *This method has already been provided for you, but its code is the same as* `runSequential()` *which needs to be edited. Remember that* `forallChunked (...) ...` *is the same as* `finish forasyncChunked (...) ...`*, you may also choose to use the* `forasyncChunked` *construct. Hint: you should have the following structure in your parallel version:* `for-forallChunked–for`.

5. 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 `numWorkerThreads()`, which is the number of workers w specified with the "`-Dhj.numWorkers`=w" option (default is $w = 8$ on SUGAR).

   (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 200,000.

   (c) `iterations` = number of iterations needed for convergence. The default value is 20,000. This default was set for expediency. For this synthetic problem, you typically might need 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.

6. You should run your program on SUGAR, to evaluate the parallelization. As before, you can compile the program as follows:

   ```
   javac -cp /users/COMP322/habanero-java-lib.jar OneDimAveraging.java
   ```

   To run the program using 8 cores, use the following command on a *compute node*:

   ```
   java -cp /users/COMP322/habanero-java-lib.jar:.  -Dhj.numWorkers=8 OneDimAveraging
   ```

7. Record in `lab_4_written.txt` the best execution times observed for the default inputs (using 8 cores) for the three variants, and then compute their ratio as the speedup. Compare your results for `runForkJoin()` and `runChunkedForkJoin()` against the running times for `runSequential()`.

4 Turning in your lab work

For each lab, you will need to turn in your work before leaving, as follows.

1. Check that all the work for today's lab is in the `lab_4` 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.

2. Use the turn-in script to submit the `lab_4` directory to your turnin directory as explained in the first handout: *turnin comp322-S14:lab_4*. 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_4.zip` *file to comp322-staff@mailman.rice.edu.*