Lab 2: Abstract Performance Metrics

Instructor: Vivek Sarkar

Resource Summary

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

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

Coursera Login: visit `http://rice.coursera.org` and select "Fundamentals of Parallel Programming"

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

Important tips and links

NOTE: It is recommended that you do the setup and execution for today's lab on your laptop computer instead of a lab computer, so that you can use your laptop for in-class activities as well. 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 "S14" instead of "s14".

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`

1 Measuring Abstract Performance Metrics with Array Sum

1. Update your `habanero-java-lib.jar` file! (`http://www.cs.rice.edu/~vs3/hjlib/habanero-java-lib.jar`). You need the latest version of the library for the programs provided today to work.
2. Download the `ArraySum1.java` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.
3. Copy the following line of code

   ```
   System.setProperty(HjSystemProperty.abstractMetrics.propertyKey(), ''true'');
   ```

   in the main program before *i*nitializeHabanero(); This line enables the generation of abstract metrics.
4. Compile and run this java program.
5. Notice the following statistics printed at the end of program execution for the default array size of 8:

(a) "`TOTAL NUMBER OF OPS DEFINED BY CALLS TO doWork()`" the total (*WORK*) in the computation in units implicitly defined by calls to doWork()

(b) "`CRITICAL PATH LENGTH OF OPS DEFINED BY CALLS TO hj.lang.perf.doWork()`", the critical path length (*CPL*) of the computation in units implicitly defined by calls to perf.doWork()

(c) "`IDEAL PARALLELISM = WORK/CPL`", the *ideal parallelism* in the computation

6. You can repeat the run for a different array size by clicking open the *Run* menu at the top and then choose *Edit Configurations*. In the popped out window, enter the size in *Program arguments* and click *OK*. Now run the program again.

   What WORK, CPL and IDEAL PARALLELISM values do you you see for different array sizes? Enter these values in a file named `lab_2_written.txt` in the `lab_2` directory. for array sizes that range across all powers of 2 up to 1024 — 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024.

2 String Search Problem

1. Download the `Search2.java` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.

2. `Search2.java` contains a sequential program to search for a substring (pattern) in a given string (text), and return the total number of occurrences found. As discussed in Lecture 4, this program has been instrumented to count each character comparison as 1 unit of work from the viewpoint of abstract performance metrics, and ignore everything else.

3. Your lab assignment is to convert it to a parallel program that produces the correct answer with a smaller critical path length (ideal parallel time) than the sequential version. You can explore alternate algorithms that reduce the critical path length further than what was discussed in the lecture.

   As discussed in the lecture, be sure that your parallel solution avoids "data races" for the shared variable `count` *e.g.,* by instead creating an array of 0/1 entries, and then computing its sum.

4. What WORK, CPL and IDEAL PARALLELISM values do you you see for the default input? Enter these values in the `lab_2_written.txt` file.

3 Array Sum Revisited

1. Download the `ArraySumLoop.java` and `ArraySumUtil.java` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`. Make sure you have the ArraySumUtil.java in the same package as ArraySumLoop.java.

2. The main difference compared to `ArraySum1.java` is that the call to `doWork()` in `ArraySumLoop.java` estimates the cost of an add as the number of significant bits in both operands. Thus, the cost depends on the values being added.

3. Look at the computeSumReduction function. Insert async and finish statements in appropriate places of the code to parallelize the summation. Think about what each line is doing in computeSumReduction. What is a step? What does doOperation() do? Can you parallelize the outer forloop that increments the step size or the inner forloop incrementing the rightIndex? Why? Feel free to ask the TAs if you have any question.

4. Verify the correctness of your implementation.

5. Again, enter WORK, CPL and IDEAL PARALLELISM values in `lab_2_written.txt` for array sizes that range across all powers of 2 up to 1024 — 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024. While it is reasonable to see higher WORK and CPL values for ArraySumLoop than ArraySum1, comment on how the IDEAL PARALLELISM for ArraySumLoop compares with that of ArraySum1.

4 Array Sum Recursive

1. Download the `ArraySumRecursive.java` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`. Make sure you have the `ArraySumUtil.java` in the same directory.
2. The main difference compared to `ArraySumLoop.java` is that it recursively calls computeSumReduction to the left and right subarray. Modify the ArraySumRecursvie.java by inserting async and finish statements.
3. Verify the correctness of your implementation.
4. Again, enter WORK, CPL and IDEAL PARALLELISM values in `lab_2_written.txt` for array sizes that range across all powers of 2 up to 1024 — 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024. Compare the CPL and WORK values for ArraySumRecursive.java with those you measured for ArraySumLoop.java. Do you notice any difference? Try explain the differences in your lab report.

5 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_2` 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. Before you leave, create a zip file of your work by changing to the parent directory for `lab_2/` and issuing the following command, "`zip -r lab_2.zip lab_2`".
3. Use the turn-in script to submit the contents of the `lab_2.zip` file as a new `lab_2` directory in your turnin directory as explained in the first handout.

NOTE: If the turnin command does not work for you, please email your `lab_2.zip` *file to comp322-staff@mailman.rice.edu.*