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

NOTE: As with Lab 1, you have the option of doing today's lab on your laptop computer or on a lab computer. If you use your laptop, the setup should work smoothly if it runs Mac OS or Linux. We will provide tips for Windows users, but we cannot guarantee that the course infrastructure will work on your Windows laptop. For Windows, you should ensure that DrHJ is launched on a standard Oracle JDK. If you're having problems using a Windows machine for your work, we recommend that you use a lab machine instead. You should have 24-hour access to the lab with your Rice ID card.

Finally, all commands below are CaSe-SeNsItIvE. For example, be sure to use "S13" instead of "s13".

1 Update your HJ Installation

Please update your HJ installation to make sure that you have the latest updates and bug fixes.

1. Download the jar file for DrHJ from `http://www.cs.rice.edu/~vs3/downloads/hj/drhj.jar`
2. A link to the above jar file can be obtained by following these links from the course web page: "HJ Info" → "HJ Download and Setup", and then searching for "Download the jar file corresponding to DrHJ"

DrHJ tip: If the source location of an error message appears unclear in the "Compiler Output" pane, click on "Console Output" and you may see a precise location in *line:column* format. For example 10:20 refers to column 20 in line 10.

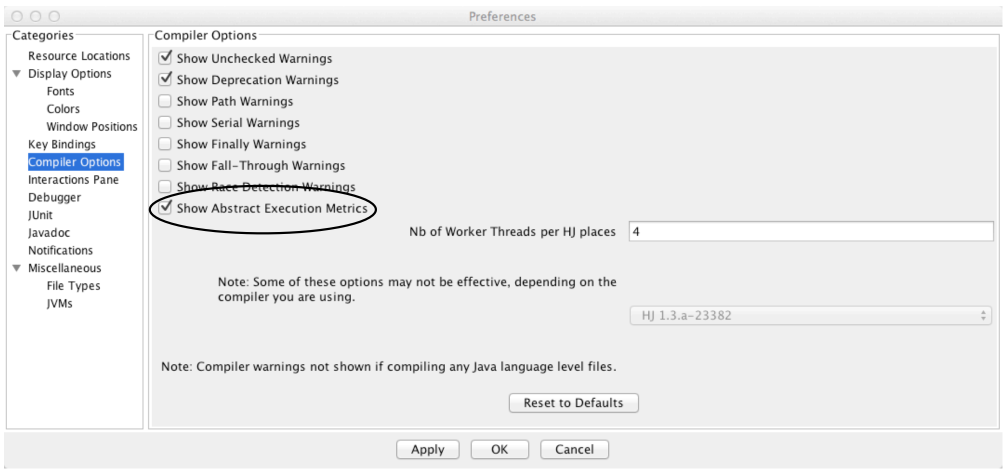
Optional note for advanced users: If you prefer to use a command-line interface instead of DrHJ to compile and run HJ programs, you can down load an HJ installation from the "HJ Download and Setup" page listed above by searching for "Download the zip file containing the HJ package" and then following the subsequent instructions. The command-line interface only works on Unix-based systems, and not on Windows. In contrast, DrHJ runs on both Unix-based systems and also on some Windows installations. We will introduce command-line interfaces for HJ to all students later in the semester.

2 Measuring Abstract Performance Metrics with Array Sum

1. Download the `ArraySum1.hj` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.
2. Compile this HJ program. Click "Compile" in DrHJ, or (if you're not using DrHJ) type the following command on the command line: *hjc ArraySum1.hj*

3. Run the program with an option to generate abstract performance metrics. Select "Show Abstract Execution Metrics" in DrHJ's Compiler Option preferences (see Figure 1), or (if you're not using DrHJ) type the following command on the command line: *hj -perf=true ArraySum1*
4. Notice the following statistics printed at the end of program execution for the default array size of 8:
   (a) "`TOTAL NUMBER OF TASKS`", the total number of async tasks created
   (b) "`TOTAL NUMBER OF OPS DEFINED BY CALLS TO hj.lang.perf.doWork()`", the total *WORK* in the computation in units implicitly defined by calls to perf.doWork()
   (c) "`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()
   (d) "`IDEAL PARALLELISM = WORK/CPL`", the *ideal parallelism* in the computation
5. You can repeat the run for a different array size by typing "`run ArraySum1` *size*" in DrHJ's Interactions Pane.

   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.



Compiler Options
[✓] Show Unchecked Warnings
[✓] Show Deprecation Warnings
[ ] Show Path Warnings
[ ] Show Serial Warnings
[ ] Show Finally Warnings
[ ] Show Fall-Through Warnings
[ ] Show Race Detection Warnings
[✓] Show Abstract Execution Metrics
Nb of Worker Threads per HJ places: 4

Note: Some of these options may not be effective, depending on the compiler you are using.

HJ 1.3.a-23382

Note: Compiler warnings not shown if compiling any Java language level files.

Figure 1: Selection of "Show Abstract Execution Metrics" in DrHJ's Compiler Option preferences

3 String Search Problem

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

4 Array Sum Revisited

1. Download the `ArraySum3.hj` file from the Code Examples column for Lab 2 in the course web page, `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.
2. The main difference compared to `ArraySum1.hj` is that the call to `doWork()` in `ArraySum3.hj` 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. 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 ArraySum3 than ArraySum1, comment on whether the IDEAL PARALLELISM is higher or lower for ArraySum3 than ArraySum1 and why that's the case.

5 Turning in your quiz and lab work

As in Lab 1, you will need to complete a quiz on Coursera and turn in your work before leaving, as follows:

1. Visit rice.coursera.org, select "Fundamentals of Parallel Programming" course, and take the Lab 2 quiz.
2. 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.
3. 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`".
4. Use the turn-in script to submit the contents of the `lab_2.zip` file as a new `lab_2` directory in your turnin repository as explained in Lab 1. You can always examine the most recent contents of your svn repository by visiting `https://svn.rice.edu/r/comp322/turnin/S13/`*your-netid*.