Lab 2: Computation Graphs, Abstract Performance Metrics

Instructor: Vivek Sarkar

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"

NOTE #1 (Optional): 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 (like CLEAR, Sugar, Mac OS), and not on Windows. In contrast, DrHJ runs on both Unix-based systems and also on some Windows installations. (For Windows, you should download a standard full JDK from Oracle to maximize the chances of DrHj working on your system.)

2 Example HJ Program with Abstract Performance Metrics

1. Download the `ArraySum1.hj` file from the Code Examples column for Lecture 3 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
   (a) "`TOTAL NUMBER OF TASKS`", the total number of async tasks created
   (b) "`TOTAL NUMBER OF OPS DEFINED BY CALLS TO hj.lang.perf.addLocalOps()`", the total *WORK* in the computation in units implicitly defined by calls to perf.addLocalOps()
   (c) "`CRITICAL PATH LENGTH OF OPS DEFINED BY CALLS TO hj.lang.perf.addLocalOps()`", the critical path length (*CPL*) of the computation in units implicitly defined by calls to perf.addLocalOps()
   (d) "`IDEAL SPEEDUP IN NUMBER OF OPS, (TOTAL NUMBER) / (CRIT PATH LENGTH)`", the *ideal speedup* or *parallelism* in the computation

3 Generating a Computation Graph Figure from an HJ Program Execution

The HJ compiler supports a "-dcg" option to enable automatic generation of computation graphs from dynamic executions of the HJ program. This option can be enabled by selecting the appropriate option in

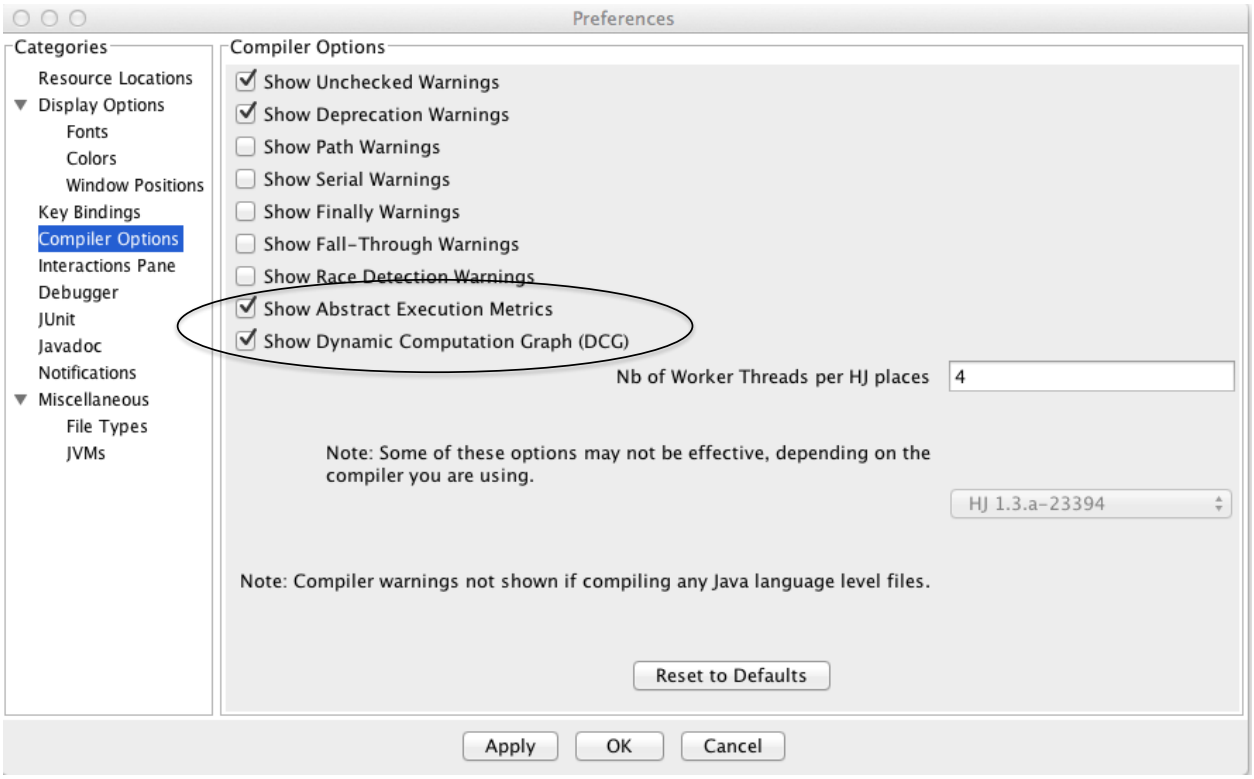

Figure 1: Selection of "Show Abstract Execution Metrics" in DrHJ's Compiler Option preferences

DrHJ's Compiler Option preferences (see Figure 1), or (if you're not using DrHJ) by typing the following command on the command line: *hjc -dcg ArraySum1*

When the HJ program is executed with this option, it will produce a file named `ArraySum1.dot`. The `.dot` file is a specification of the Computation Graph as a directed graph that can be visualized using the GraphViz package available at `http://www.graphviz.org/`. The `dot` command can also be used on the command line on CLEAR systems. Once installed, you can either use theGraphViz program to examine the `.dot` file, or issue the following command on the command line: *dot -Tpdf ArraySum1 -o ArraySum1.pdf*

Examine the Computation Graph figure obtained for n=8 (the default value for ArraySum1) and match each node in the graph with one or more statements from ArraySum1.hj.

4 Abstract Performance Metrics for Substring Search Problem

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

2. `Search.hj` contains a sequential program to search for a substring (pattern) in a given string (text). 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.