Lab 2: Abstract Metrics

Instructor: Vivek Sarkar, Co-Instructor: Shams Imam

Course Wiki: http://comp322.rice.edu

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

Goals for this lab

- Understand abstract metrics
- Understand actual speedup metrics
- One HJlib API: `doWork`

1 Setup

As in lab 1, download the `lab_2` project on your machine using one of the following methods. Note: The URL for lab 2 is `https://svn.rice.edu/r/comp322/turnin/S16/NETID/lab_2/` where you will need to replace `NETID` with your net id.

1. Download the project using the IntelliJ support for Subversion (Instructions with Images).
2. Download the project using the command line (Demo Video).
3. If you do not have subversion set up on your machine, you can download the lab_2.zip file and manually set up the project on IntelliJ. Then you can use the `svn commit` command to submit your changes.

Note that if you have the `-javaagent` set up in your IntelliJ run configuration, you can use standard IntelliJ debugging features (e.g. breakpoints) to debug your code.

2 ReciprocalArraySum Program Revisited

This week we will revisit the simple two-way parallel array sum program introduced last week and in the Demonstration Video for Topic 1.1. You will edit the `ReciprocalArraySum.java` program provided in your svn repository for this exercise (do not use the version from the video lecture). There are `TODO`s in the `ReciprocalArraySum.java` file guiding you on where to place your edits.

- The goal of this exercise is to create an array of `N` random doubles, and compute the sum of their reciprocals in several ways, then comparing the benefits and disadvantages of each:
  - Sequentially in method seqArraySum().
  - In parallel using **two** asyncs in method parArraySum_2asyncs(). You have already written code almost exactly like this in Lab 1 last week! Remember to add the calls to doWork() as seen in the seqArraySum() method to keep track of abstract metrics. For the default input size, our solution achieved an ideal parallelism of *just under* 2.
  - In parallel using **four** asyncs in method parArraySum_4asyncs(). You are essentially creating a version of parArraySum_2asyncs that uses 4 asyncs instead of 2. Think about the following questions: How do you want to split up the work among the 4 tasks? Equally? Is this the best way? For the default input size, our solution achieved an ideal parallelism of *just under* 4.

  - Lastly, in parallel using **eight** asyncs in method parArraySum_8asyncs(). You are essentially creating a version of parArraySum_2asyncs that uses 8 asyncs instead of 2. Think about the following questions: Do you really want to have to manually create 8 asyncs manually? Is there a better way you could write this function? Remember that copying and pasting code is generally discouraged. For the default input size, our solution achieved an ideal parallelism of *just under* 8.

- Compile and run the program in IntelliJ to ensure that the program runs correctly without your changes. Follow the instructions for "Step 4: Your first project" in https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124. If you're not using IntelliJ, you can do this by running the `mvn clean compile exec:exec -Preciprocal` command as specified in the README file.

  Be sure you **run the ReciprocalArraySumTest** file, not the ReciprocalArraySum file.

- Compare the abstract metric results and the actual speedup metric results and be able to explain the discrepancies before leaving lab. Note that the actual speedups depend on the input array size, which is 10^6 for today's lab, as well as the characteristics of your laptop.

3 Parallelizing computing Combinations

A *combination* is an arrangement of all or part of a set of objects, with regard to the order of the arrangement. You are provided a sequential solution to find the number of combinations that is possible from a list of `n` elements in `Combinations.java`. The provided solution relies on the use of a functional style of lists (`FunctionalList.java`).

Your task in this lab is to parallelize the provided solution using only the `finish` and `async` constructs. There is a `TODO` in the `Combinations.java` file guiding you on where to place your edits. Try to maximize the amount of parallelism in the computation and inspect the parallelism you are able to exploit by viewing the results computed by abstract metrics. For the default input of size 8, our solution achieved an ideal parallelism of over 5.5. You can verify the correctness of your solution by running the `CombinationsTest` test file.

NOTE: You may not modify the `FunctionalList.java` *file for this lab (unless you need to correct checkstyle errors).*

4 Demonstrating and submitting in your lab work

For this lab, you will need to demonstrate and submit your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab. They will want to see your files submitted to Subversion in your web browser, the passing unit test on your laptop, and the passing result in the autograder UI.

   For both the programs (`ReciprocalArraySum` and `Combinations`) you wrote today:

   - What is the trend in the abstract metrics as the number of `async`s increases?
   - What is the trend in the actual speedup metrics as the number of `async`s increases?
   - Is there a dependency between these two trends? If so, what is this dependency? What might be causing it? How might you go about resolving this slowdown problem?

2. For the `Combinations` program explain the critical path you obtain from the parallel version of your program. It might help to inspect the work done and critical path length of your programs with input values of 0, 1, 2, and 3.

3. Check that all the work for today's lab is in the `lab_2` directory. Enter the answers to the above questions in a file named `lab_2_written.txt` in the `lab_2` directory. If not, make a copy of any missing files/folders there.

4. Submit all your changes in the `lab_2` directory using subversion. Remember to explicitly add any new files (e.g. your report or any new Java files) you created into the subversion repository.