Lab 4: Loop-Level Parallelism

Instructor: Vivek Sarkar

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

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

Goals for this lab

- Understand parallel for loops
- Experiment with chunking techniques
- Get familiar with Computation Graph visualization with HJ-Viz
- Two HJ-lib APIs: `forall` , `forallChunked`

Lab Projects

The Maven project for this lab is located in the following svn repository:

- `https://svn.rice.edu/r/comp322/turnin/S15/`*`NETID`*`/lab_4`

Please use the subversion command-line client to checkout the project into appropriate directories locally. For example, you can use the following commands from a shell:

```
$ cd ~/comp322
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_4_forall lab_4_forall
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_4_hjviz lab_4_hjviz
```

1 Getting Familiar with forall

1.1 Parallel Image Manipulation

In this exercise, you will be altering the pixel values of .jpg images. The result will be a negative copy of the original. You will utilize loop-level parallelism to alter multiple sections of an image simultaneously.

In this exercise you are given the sequential implementation for the image converter. You will parallelize the `ImageNegative` code by completing the imageNegativePar method where specified. It will be helpful to refer to the sequential implementation for the syntax of the operation. Do not destructively modify the input image.

In order to run this file, you will need to include the name of the file as a program argument in the following format: `filename.jpg` The files `test.jpg` and `test2.jpg` have been included for your convenience. However, feel free to run the program using your own .jpg files. You will need to add your image to the `src/main/resources` folder.

While completing your solution, keep in mind the trade off between ideal parallelism and real execution time. Spawning asynchronous tasks can incur expensive overhead. Be careful with the number of tasks you generate.

1.2 Verifying your Solution

Run your completed solution file, and verify the correctness of your solution by examining the output images generated. You may run the unit tests (`mvn clean test`) to determine if your solution has worked. Keep in mind that the test may fail due to speed, even if your solution is functionally correct.

2 Improving your Solution with Chunking

Your final goal in this lab is to improve the actual performance of your program to the greatest extent possible. This can be done through chunking. Since creating asynchronous tasks is expensive, it may be helpful to divide your input into several chunks, and then process the contents of each chunk sequentially, with each chunk being processed asynchronously. The `forallChunked` statement is used for this purpose. You may either specify the size of each chunk, or allow `HJLib` to automatically divide your input based on the number of worker threads available for use.

Due to system variability, the speedup required to pass the unit test is low. However it is strongly encouraged you make your solution as efficient and elegant as possible.

Computation Graph visualization with HJ-Viz

HJ-Viz, which generates interactive Computation Graphs (CGs) of parallel programs by analyzing event logs. It also incorporate Abstract Execution Metrics (AEM) in the visualization. HJ-Viz highlights the program's critical paths and displays the amount of work performed in each step of computation based on the collected AEM.

In this part of the lab, your goal is familiarize yourself with the use of HJ-Viz. You will use the maven project inside the `lab_4_hjviz` maven project for this part of the lab. The project should contain the following java files in the `src/main/java/edu/rice/comp322` folder:

- `ReciprocalArraySumFutures.java`
- `ArraySum4.java`
- `MergeSort.java`
- `Puzzle.java`
- Lab4Util.java

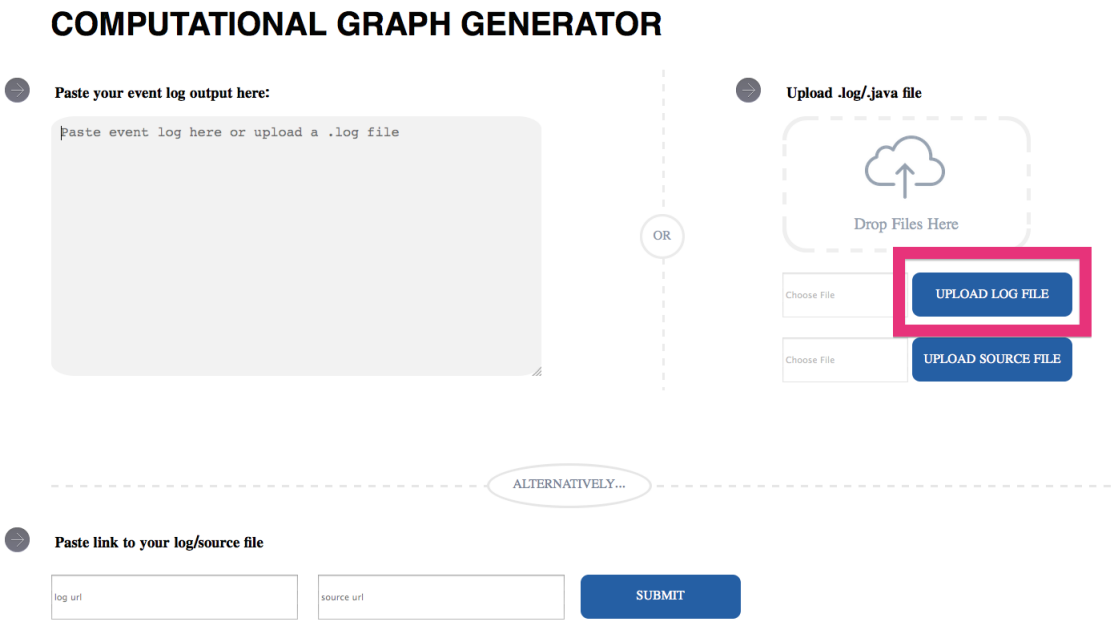
2.1 Viewing CGs of existing HJlib programs

`ReciprocalArraySum.java`, `ArraySum4.java`, and `MergeSort.java` contain correct HJ-lib programs. They are currently configured to run on small input sizes and generate event logs files for you to use. Please run the following commands to generate the event log files:

```
$ mvn clean compile exec:exec -Preciprocal
$ mvn clean compile exec:exec -Parraysum4
$ mvn clean compile exec:exec -Pmergesort
```

Running these commands should generate *.log files which can be used by HJ-Viz in visualization. Please remember the names of the files generated as output, these files will be used later as inputs to HJ-Viz for CG visualization.

Figure 1: Entry page of HJ-Viz.



HJ-Viz relies on the use of GraphViz (`http://www.graphviz.org/Download..php`), please install GraphViz on your machines before proceeding. Once you have GraphViz installed, we need to run the HJ-Viz web server on your machine locally. This can be done by running the following command:

```
$ mvn clean compile exec:exec -Pserver
...
[INFO] --- exec-maven-plugin:1.3.2:exec (default-cli) @ lab4-hjviz ---
* Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

This messages means that HJ-Viz successfully started on your machine and can be viewed by clicking the following url: `http://127.0.0.1:8000/`. Once you click on the url, a page as follows will load as seen in fig. 1.

Use the "upload log file" button to upload your log files generated in the previous steps[1]. For the `ReciprocalArraySum` example, you should see something as in fig. 2.

2.2 CG Puzzle

Your final task in this lab is to write a HJ-lib program that generated as CG shown in fig. 3. You will edit the `edu.rice.comp322.Puzzle#puzzle()` method for this task.

Turning in your lab work

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

1. Show your work to an instructor or TA to get credit for this lab (as in COMP 215). Be prepared to explain the lab at a high level, as well as answer the following questions:

[1] You might need to refresh the page as HJ-Viz intermittently clears its session cache.

Figure 2: CG of ReciprocalArraySum.

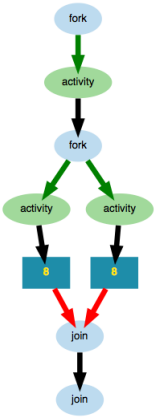
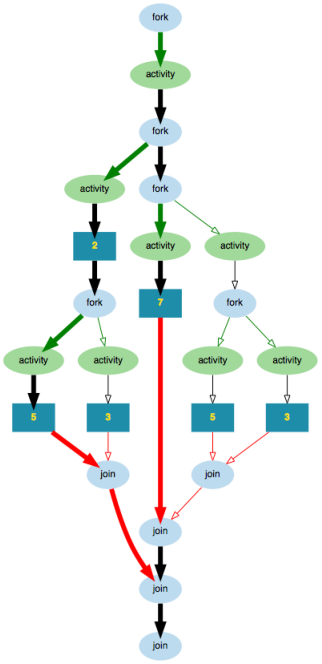


Figure 3: CG of Puzzle.



- What was your strategy in rewriting the provided sequential image manipulation as a parallel one?
- How did you determine chunk size? How did various chunk sizes affect your speedup?
- What are the drawbacks to your solution? What improvements can be made?

2. 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.

3. Use the turn-in script to submit the `lab_4` directory to your turnin directory as explained in the first handout: *turnin comp322-S15: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.*