Lab 5: Finish-Async Parallel Programming with Abstract Metrics

Instructor: Mack Joyner

Course wiki: `http://comp322.rice.edu`

Staff Email: comp322-staff@rice.edu

Goals for this lab

- Three HJ-lib APIs: `launchHabaneroApp`, `async`, and `finish`.
- Abstract metrics with calls to `doWork()`.

Downloads

As with lab 1, the provided template project is accessible through your private GitHub classroom repo at: `https://classroom.github.com/a/Emdg4ueR`

For instructions on checking out this repo through IntelliJ or through the command-line, please see the Lab 1 handout. The below instructions will assume that you have already checked out the lab6 folder, and that you have imported it as a Maven Project if you are using IntelliJ.

1 Getting Familiar with Finish-Asyncs

You can think of asyncs as a future without a return value. Like a future, the logic associated with a async takes place asynchronously, not necessarily when the async is created. Different from a future, however, is that async logic doesn't return a value, which was of course only accessible after the future has completed. A `finish` will block all async tasks inside of the `finish` until the tasks have completed.

1.1 Pascal's Triangle With Finish-Async

Pascal's Triangle is a recursive algorithm that can be visualized as follows. In the initial step, we create a triangle of integers and initialize all border entries to one. We say that this triangle has `N` rows, and that each row has `K` columns (where `N` is fixed but `K` varies by row, where `K` for row `n` is `n + 1`). Rows and columns are numbered starting at zero. Figure 1 depicts an initialized Pascal's Triangle.

```
          1
        1   1
      1       1
    1           1
  1               1
1                   1
```

Figure 1: An initialized Pascal's triangle for N = 6.

To fill in each empty cell of the triangle, we sum the values to its top left and top right. For example, to compute the the element for `n = 2` and `k = 1` we would sum the values stored at (1, 0) and (1, 1). Figure 2 depicts a Pascal's Triangle with element (2, 1) filled in.

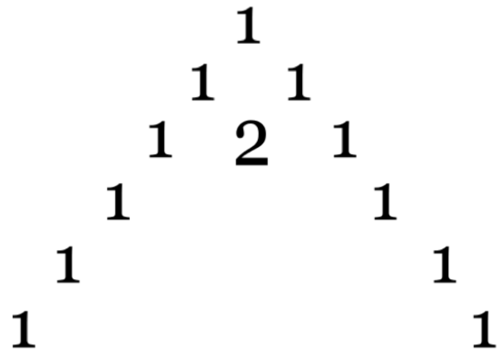

Figure 2: An example of filling in element (2, 1) for a Pascal's triangle with N = 6.

Applying this algorithm recursively by row would produce the complete triangle in Figure 3.

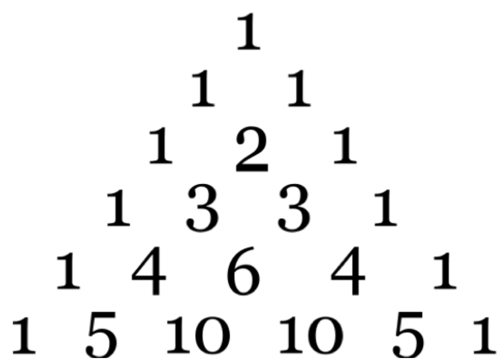

Figure 3: A complete Pascal's Triangle for N = 6.

In this lab, you will need to edit `PascalsTriangle.java` to produce a correct parallel solution using finish and async. In `PascalsTriangle.java` you will find a reference sequential version which you can use `finish` and the `async` API to parallelize. You must ensure that a call do `doWork(1)` is made for each addition of two parent nodes to calculate a child node's value. Running `PascalsTriangleCorrectnessTest.java` will verify the correctness and abstract performance of your solution.

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 checked off for this lab during lab or office hours by Monday, Feb 20th at 3pm. They will want to see your passing unit tests on your laptop.
2. Commit and push your code to GitHub.