Homework 1: due by 5pm on Friday, January 31, 2014 (Total: 100 points)

Instructor: Vivek Sarkar

All homeworks should be submitted in a directory named hw_1 using the turn-in script. In case of problems using the script, you should email a zip file containing the directory to comp322-staff@mailman.rice.edu before the deadline. See course wiki for late submission penalties.

Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone elses work as your own. If you use any material from external sources, you must provide proper attribution.

1 Written Assignments (50 points total)

Please submit your solutions to the written assignments in either a plain text file named ***hw_1_written.txt*** *or a PDF file named* ***hw_1_written.pdf*** *in the hw_1 directory.*

1.1 Finish Synchronization (20 points)

Consider the sequential and incorrect parallel versions of the HJ code fragment included below. *Your task is to only insert finish statements in the incorrect parallel version so as to make it correct i.e., to ensure that the parallel version computes the same result as the sequential version, while maximizing the potential parallelism.*

```
// SEQUENTIAL VERSION:
  for ( p = first; p != null; p = p.next) p.x = p.y + p.z;
  for ( p = first; p != null; p = p.next) sum += p.x;

// INCORRECT PARALLEL VERSION:
  for ( p = first; p != null; p = p.next) async p.x = p.y + p.z;
  for ( p = first; p != null; p = p.next) sum += p.x;
```

1.2 Amdahl's Law (30 points)

In Lecture 4 (Topic 1.5), you will learn the following statement of Amdahls Law:

> If q $\leq$ 1 is the fraction of WORK in a parallel program that must be executed sequentially, then the best speedup that can be obtained for that program, even with an unbounded number of processors, is Speedup $\leq$ 1/q.

Now, consider the following generalization of Amdahls Law. Let q1 be the fraction of WORK in a parallel program that must be executed sequentially, and q2 be the fraction of WORK that can use at most 2 processors. Assume that the fractions of WORK represented by q1 and q2 are disjoint. Your assignment is to provide an upper bound on the Speedup as a function of q1 and q2, and justify why it is a correct upper bound. (Hint: to check your answer, consider the cases when q1=0 or q2=0.)

2 Programming Assignment (50 points)

2.1 Habanero-Java Library (HJ-lib) Setup

See Lab 1 for instructions on HJ-lib installation for use in this homework, and Lecture 3 and Section 1.4.1 of the Module 1 handout for information on abstract execution metrics. To avoid any possible discrepancies with the use of different versions of HJ-lib, **please download and only use the current HJ-lib jar file: habanero-java-lib-201401162331.jar for this assignment.**

Download Url: `http://www.cs.rice.edu/~vs3/hjlib/code/habanero-java-lib-201401162331.jar`

2.2 Parallel Quicksort (50 points)

Quicksort is a classical sequential sorting algorithm introduced by C.A.R. Hoare in 1961, and is still very much in use today. We have provided three source files with the homework document in the course wiki — `QuicksortUtil.java`, `QuicksortSeq.java`, and `QuicksortPar.java` . `QuicksortUtil.java` contains the main() method with calls to `initializeHabanero()` and `finalizeHabanero()`. As discussed in Lecture 3, it also enables an option to generate abstract performance metrics by inserting the method call

```
System.setProperty(HjSystemProperty.abstractMetrics.propertyKey(), "true");
```

before the call to `initializeHabanero()`. The main() method calls `QuicksortSeq.quicksort()` followed by `QuicksortPar.quicksort()`, and reports abstract metrics for each of them by calling `dumpStatistics()` which prints out metrics as follows:

```
#### START OF ABSTRACT EXECUTION STATISTICS ####
    WORK = 45830 [total number of ops defined by calls to HJ.doWork()]
    CPL = 45830 [critical path length of ops defined by calls to HJ.doWork()]
    IDEAL PARALLELISM = WORK/CPL = 1.0
#### END OF ABSTRACT EXECUTION STATISTICS ####
```

For this assignment, the only operations performed on the array are methods `exchange()` and `compareTo()` defined in `QuicksortUtil.java`. For abstract metrics, we assume that one call to either method equals 1 unit of work, hence the call to `doWork(1)` in each method. All other operations are ignored *i.e.,* are assumed to have negligible/zero cost.

`QuicksortSeq.java` contains a sequential implementation of the Quicksort algorithm, so it will always result in identical `WORK` and `CPL` metrics. `QuicksortPar.java` currently contains a copy of `QuicksortSeq.java`. *Your assignment is to replace the sequential code in `QuicksortPar.java` by a correct parallel program that always performs the same WORK as the sequential version, but results in the smallest possible CPL value.* A correct parallel program will generate the same output as the sequential version and will also not exhibit any data races. The parallelism in your solution should be expressed using only `async`, `finish`, and/or `future` constructs.

Your edits should be restricted to the `QuicksortPar.java` file. Both `QuicksortSeq.java` and `QuicksortUtil.java` should remain unchanged. We will execute your `QuicksortPar.java` file with the original `QuicksortSeq.java` and `QuicksortUtil.java` files when grading your homework.

Your submission should include the following in the hw_1 directory:

1. (25 points) A complete parallel solution for Quicksort as outlined above in a modified `QuicksortPar.java` file. We will only evaluate its performance using abstract metrics, and not its actual execution time. All code should include basic documentation for each method in each class, as you've been taught in prior CS classes.

   See Section 1.6 in the Module 1 handout, as well as the lecture and demonstration videos for topic 1.6, for technical details on parallel variants of the Quicksort algorithm. Approach 1 is the simplest

of the three parallelization approaches discussed in the Module 1 handout and the video lectures, but Approach 3 in the Module 1 handout will give the smallest critical path length. (Note that "forall" in Approach 2 and Approach 3 is simply a shorthand for a parallel for-async loop structure.)

With Approach 1, you will get a maximum of 20 out of 25 points for this part of the homework. However, you can get all 25 points with an Approach 3 solution. It is recommended that you complete the entire homework with Approach 1 first, and then explore Approach 3 if time permits.

2. (15 points) A report file formatted either as a plain text file named `hw_1_report.txt` or a PDF file named `hw_1_report.pdf` in the hw_1 directory. The report should summarize the design of your parallel solution, and explain why you believe that your implementation is correct, data-race-free, and maximally parallel (to the best of your effort).

3. (10 points) The report file should also include test output for sorting arrays of size n = 1000. The test output should include both the result value and the WORK, CPL, and IDEAL SPEEDUP (= WORK/CPL) from each run.