

# Homework 1: due by 12noon on Friday, January 29, 2016

(Total: 100 points)

Instructor: Vivek Sarkar, Co-Instructor: Shams Imam.

All homeworks should be submitted in a directory named `hw_1` in your svn repository for this course. In case of problems committing your files, please contact the teaching staff at [comp322-staff@mailman.rice.edu](mailto:comp322-staff@mailman.rice.edu) before the deadline to get help resolving for your issues. No late submissions will be accepted unless you are using your slip days.

The slip day policy for COMP 322 is similar to that of COMP 321. All students will be given 3 slip days to use throughout the semester. When you use a slip day, you will receive up to 24 additional hours to complete the assignment. You may use these slip days in any way you see fit (3 days on one assignment, 1 day each on 3 assignments, etc.). If you use slip days, you must submit a `README.txt` file in your svn homework folder before the actual submission deadline indicating the number of slip days that you plan to use.

Other than slip days, no extensions will be given unless there are exceptional circumstances (such as severe sickness, not because you have too much other work). Such extensions must be requested and approved by the instructor (via e-mail, phone, or in person) before the due date for the assignment. Last minute requests are likely to be denied.

If you see an ambiguity or inconsistency in a question, please seek a clarification on Piazza or from the teaching staff. If it is not resolved through those channels, you should state the ambiguity/inconsistency that you see, as well as any assumptions that you make to resolve it.

*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 else's work as your own. If you use any material from external sources, you must provide proper attribution.*

## 1 Written Assignments (50 points total)

Submit your solutions to the written assignments as a PDF file named `hw_1_written.pdf` in the `hw_1` directory. Please note that you be penalized 10 points if you misplace the file in some other folder or if you submit the report in some other format.

### 1.1 Finish Synchronization (20 points)

Consider the sequential and incorrect parallel versions of the pseudocode fragment included below. *Your task is to only insert finish statements in the incorrect parallel version so as to ensure that the parallel version contains no data races, computes the same result as the sequential version, and maximizes the potential parallelism. Explain your answer.*

```
// 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 ( pp = first; pp != null; pp = pp.next) {
    T p = pp; // Assume that p and pp point to objects of type T
    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  $\text{Speedup} \leq 1/q$ .

Now, consider the following generalization of Amdahls Law. Let  $q_1$  be the fraction of WORK in a parallel program that must be executed sequentially,  $q_2$  be the fraction of WORK that can use at most 2 processors, and  $(1 - q_1 - q_2)$  the fraction of WORK that can use an unbounded number of processors. Assume that the fractions of WORK represented by  $q_1$ ,  $q_2$ , and  $(1 - q_1 - q_2)$  are disjoint, and cannot be overlapped with each other in a parallel execution. Your assignment is to provide an upper bound on the Speedup as a function of  $q_1$  and  $q_2$ , and justify why it is a correct upper bound. (Hint: to check your answer, consider the cases when  $q_1=0$  or  $q_2=0$ .)

## 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 1.3 of the [Module 1](#) handout for information on abstract execution metrics. In addition, the `hw_1` project includes a README file which includes helpful Maven commands to compile and run your files.

### 2.2 Parallel Sort (50 points)

In this homework, we have provided you with sequential implementations of different sorting algorithms. Each of these sorting algorithms have been implemented in a file of their own, e.g. `QuickSort.java`, `BitonicSort.java`, `MergeSort.java`, etc. As in `lab_1`, the homework project will be available in your svn repository at [https://svn.rice.edu/r/comp322/turnin/S16/your-netid/hw\\_1](https://svn.rice.edu/r/comp322/turnin/S16/your-netid/hw_1).

*Your assignment is to write a correct parallel version of any sort algorithm that you choose by overriding the `parSort()` method. The goal is to obtain the smallest possible CPL value that you can for the given input, as measured using abstract execution metrics. A secondary goal is to not increase the WORK value significantly when doing so, but some increase is fine. Your edits should be restricted to the file for that given sort algorithm and the `sortInstance()` method in `Homework1.java`. A correct parallel program should generate the same output as the sequential version, and should not exhibit any data races. The parallelism in your solution should be expressed using only `async`, `finish`, and/or `future` constructs. It should pass the unit tests provided, and other tests that the teaching staff will use while grading.*

For the abstract metrics in this assignment, we only count 1 unit of work for each call to `compareTo()`, and ignore the cost of everything else. While this seems idealistic, it is a reasonable assumption when sorting is performed on objects with large keys.

Your submission should include the following in the `hw_1` directory:

1. (25 points) A complete parallel solution for a sorting algorithm of your choice in a modified Java file. For example, if you chose to parallelize Merge Sort, you need to submit `MergeSort.java` to the svn repository. In addition, you will need to edit the `sortInstance()` method in `Homework1.java` to return an instance of `MergeSort`. We will only evaluate its performance using abstract metrics, and not its actual execution time.

15 points will be allocated based on the ideal parallelism that you achieve. You will get the full 15 points if you achieve a CPL of  $(\log_2 n)^2$  for an array of  $n$  elements, or better.

10 points will be allocated for coding style and documentation. We have provided the basic checkstyle rules as part of your Maven project. At a minimum, all code should include basic documentation for

each method in each class. You are also welcome to add additional unit tests to test corner cases and ensure the correctness of your implementations.

2. (15 points) A report file formatted as a PDF file named `hw_1_report.pdf` in the `hw_1` directory. The report should contain the following:
  - (a) A summary of your parallel algorithm. You should write a few sentences describing the approach and the algorithm.
  - (b) An explanation as to why you believe that your implementation is correct and data-race-free
  - (c) An explanation of what values of WORK and CPL (as formulae of  $n$ ) you expect to see from your algorithm.
3. (10 points) The report file should also include test output for sorting an array of size  $n = 1024$  (i.e. the unit test name `testRandomDataInput1K`). The test output should include the WORK, CPL, and IDEAL PARALLELISM (= WORK/CPL) values from each run.