Homework 1: due by 11:59pm on Wednesday, January 29, 2020 (Total: 100 points)

Mackale Joyner

Commit all work to the svn repo at `https://svn.rice.edu/r/comp322/turnin/S20/`*your-netid*`/hw_1` that we created for you. In case of problems committing your files, please contact the teaching staff at comp322-staff@mailman.rice.edu before the deadline to get help resolving your issues.

Your solution to the written assignment should be submitted as a PDF file named `hw_1_written.pdf` in the hw_1 directory. This is important — you will be penalized 10 points if you place the file in some other folder or with some other name. The PDF file can be created however you choose. If you scan handwritten text, make sure that the writing is clearly legible in the scanned copy. Your solution to the programming assignment should be submitted in the appropriate location in the hw_1 directory.

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 plan to use a slip day, you need to say so in an svn committed README.txt file before the deadline. You should specifically mention how many slip days you plan to use. The README.txt file should be placed in the top level directory (e.g. hw_1). 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 ambiguity or inconsistency in a question, please seek clarification on Piazza (remember not to share homework solutions in public posts) 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 instructors, 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)

As mentioned earlier, your solution to the written assignment should be submitted as a PDF file named `hw_1_written.pdf` in the hw_1 directory.

1.1 Parallelizing the *IsKColorable* Algorithm for Undirected Graphs (25 points)

Consider the pseudo-code of the **IsKColorable** algorithm for undirected graphs shown in Algorithm 1 that you learned in COMP 182. *Your assignment is as follows:*

1. *(10 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything else has zero cost. Create a parallel version of this algorithm using finish and async statements so as to maximize parallelism, while ensuring that the parallel version always computes the same result as the sequential version. You may change data structures (e.g., replace a scalar variable by an array) if needed.*
2. *(5 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything*

else has zero cost. Is it possible for your parallel algorithm to have a **larger** *value of WORK than the sequential version for a given set of inputs? If so, describe an example when this can happen. If not, explain why not.*

3. *(5 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything else has zero cost. Is it possible for your parallel algorithm to have a* **smaller** *value of WORK than the sequential version for a given set of inputs? If so, describe an example when this can happen. If not, explain why not.*

4. *(5 points) Is it possible for your program to exhibit a data race according to the definition introduced in class? If so, would you consider that data race to be benign?*

Algorithm 1: IsKColorable

Input: Graph $g = (V, E)$ and $k \in \mathbb{N}$.
Output: $\exists f : V \to [k], \forall \{u, v\} \in E, f(u) \neq f(v)$.

```
1 foreach Assignment f of a value in [k] to each node in V do
2     colorable ← True;
3     foreach {u, v} ∈ E do
4         if f(u) = f(v) then
5             colorable ← False;
6             break;
7     if colorable = True then
8         return True;
9 return False;
```

1.2 Amdahl's Law (25 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 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. 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 is as follows:*

1. *(10 points) Provide the best possible (smallest) upper bound on the Speedup as a function of q1 and q2.*

2. *(15 points) Explain your answer, and justify why it is a correct upper bound. Use the cases when $q_1 = 0$, $q_2 = 0$, $q_1 = 1$ or $q_2 = 1$ to explain why your bound is correct.*

Hints:

- As with Amdahl's Law, your answer should not include the number of processors, P. It should be an upper bound that applies to all values of P.
- To check your answer, consider the cases when q_1 or q_2 are equal to 0 or 1.

2 Programming Assignment (50 points)

2.1 Habanero-Java Library (HJ-lib) Setup

See the Lab 1 handout for instructions on HJ-lib installation for use in this homework, and Lecture 3 for information on abstract execution metrics.

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/S20/`*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` and `finish` constructs. It should pass the unit tests provided, and other tests that the teaching staff may use while grading.

For the abstract metrics in this assignment, we only count 1 unit of work for each call to `compareTo()`. We'll 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 ensure `MergeSort.java` is included in your submission. 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 $O((log_2 n)^2)$ or better for an array of n elements, for all sorting algorithms except MergeSort. If you decide to parallelize MergeSort, you need to achieve CPL of $O(n)$ or better to get the full credit.

   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 implementation.

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.

3 Submitting Your Assignment

You will need to add and commit all work to the svn repository at `https://svn.rice.edu/r/comp322/turnin/S20/`*yournetid*`/hw_1`. Please open a browser, navigate to the url, and verify that you successfully committed your homework. Don't forget to add a README.txt file if you plan to use slip days.