Homework 2: due by 5pm on Wednesday, February 6, 2013 (Total: 100 points)

Instructor: Vivek Sarkar

All homeworks should be submitted in a directory named hw_2 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_2_written.txt` *or a PDF file named* `hw_2_written.pdf` *in the hw_2 directory.*

1.1 Amdahl's Law (20 points)

In Lecture 4, you learned 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.)

1.2 Parallel Prefix Sum (30 points)

Lectures 8 and 9 will introduce the Parallel Prefix Sum algorithm, which computes partial sums for an array of n elements with $CPL(n) = O(\log n)$. The algorithm will be described as an upward sweep followed by a downward sweep.

1. For this part, your assignment is to provide pseudocode for the Parallel Prefix Sum problem using whatever HJ constructs you choose for parallelism. For simplicity, you can assume that n is restricted to be a power of 2. This is a written assignment, not a programming assignment *i.e.,* you do not need to compile and run your code. It is also acceptable if details that are not central to the parallel structure of the code are written in English prose rather than real code. Your primary task is to specify how the upward and downward sweeps can be implemented using HJ parallel constructs. You may choose to provide an iterative or a recursive parallel solution.
2. For a given n, what is the minimum number of processors, $P(n)$, that can be used to achieve a speedup of $O(n/\log n)$ relative to the $O(n)$ sequential algorithm? As in the lectures, assume that the execution time of this algorithm on $P(n)$ processors can be approximated as $WORK(n)/P(n) + CPL(n)$.

2 Programming Assignment (50 points)

2.1 Setup

You should download four files posted alongside the HW2 link in the course web site: `GeneralizedReduce.hj`, `GeneralizedReduceApp.hj`, `SumReduction.hj`, and `TestSumReduction.hj`.

IMPORTANT: Among these files, you should edit `GeneralizedReduce.hj`, and leave the other files unchanged. Further, you should not modify any of the existing declarations in `GeneralizedReduce.hj`. Making additional modifications will make it difficult for the teaching staff to run automated scripts to evaluate your code, and may result in penalties to your grade.

2.2 Generalized Reduce (50 points)

The goal of this assignment is to implement a `GeneralizedReduce` *class in HJ to perform a user-specified reduction on an array of Objects.* A skeleton for this class is available in the `GeneralizedReduce.hj` file.

Your `GeneralizedReduce` class should support any class that implements the `GeneralizedReduceApp` interface with `init()` and `combine()` methods to define a reduction operation. As an example, class `SumReduction` implements the `GeneralizedReduceApp` interface to specify a sum reduction. Further, `TestSumReduction.hj` contains a test harness for `SumReduction`. Since methods in `GeneralizedReduceApp` use objects as parameters, it is necessary to convert between primitive int values and Integer objects in `SumReduction`[1].

You should use abstract metrics to evaluate the parallelism in your solution. Specifically, any reduction client should include a call to `perf.doWork(1)` for each call to the `combine()` method, as in `SumReduction.hj`. Thus, the total WORK and CPL (critical path length) for your HJ program executions will be evaluated assuming that each call to `combine()` takes one unit of time. You will not be penalized if the actual execution time of your parallel program is large due to overheads, so long as the abstract metrics are correct. Note that the abstract metrics are independent of the computer that you run your HJ program on.

Your submission should include the following in the lab_2 directory; all code should include basic documentation for each method in each class:

1. (20 points) A complete parallel implementation of the `GeneralizedReduce` class in HJ. You should aim for maximum parallelism, assuming that each call to `combine()` just does 1 unit of work. The implementation can use `async`, `finish` and/or `future` constructs, but not finish accumulators. Test your implementation with the `TestSumReduction` test harness.

2. (15 points) A complete implementation of new `MaxReduction` and `TestMaxReduction` classes in HJ to test your `GeneralizedReduce` implementation for a special max-index reduction that returns both the max value and its index in an input integer array. If the max value has multiple occurrences in the input array, then you should return the index of the first occurrence. For example, if the input array is [(5,0), (1,1), (-8,2), (5,3), (0,4)] where each element consists of a value paired with its index in the array, the output should consist of (5, 0), since 5 is the largest value in the array and its first occurrence is at index 0.

3. (15 points) A report file formatted either as a plain text file named `hw_2_report.txt` or a PDF file named `hw_2_report.pdf` in the hw_2 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. It should also include the following:

   (a) Abstract metrics obtained from executing both TestSumReduction and TestMaxReduction with randomly-generated arrays of size n = 1, 10, 100, 1000. The report text should include the the WORK, CPL, and IDEAL SPEEDUP (= WORK/CPL) from each run.

[1] If you prefer, you can replace Object's by generic type parameters.

(b) Data race detection output for TestSumReduction executed on an input size of $n = 100$. (Recall that both abstract metrics and data race detection options cannot be enabled at the same time in DrHJ.)