Exercises for lab 5 of CS2101a

Instructor: Marc Moreno Maza, TA: Xiaohui Chen

Thursday 10 October 2013

1 Exercise 1

The goal of this exercise is, for an input array `A` of integer numbers, compute the maximum value and the minimum value of an entry of `A`. To learn more about this problem and an algorithmic solution, you should listen to the following video lecture

`http://www.youtube.com/watch?v=dEyRl9Tj83g`

1. Write a `Julia` function that computes the maximum element and minimum element of an array `A` using the divide and conquer algorithm described in this video lecture.
2. Write a parallel `Julia` function for the same problem of computing the maximum and minimum value of an entry of `A`. You should use the `Julia` parallel constructs presented in class.

2 Exercise 2

The goal of this exercise is to obtain a parallel `Julia` implementation of the merge-sort algorithm. You can review this algorithm at

`http://en.wikipedia.org/wiki/Merge_sort`

You will find there several ways of presenting this algorithm. The one of the *Top-down implementation* section is similar to that proposed as solution of Lab 3. They both overwritten the input array. In other words, their sorting process can be seen as in place [1] To learn more about this idea of *working-in-place*, read the page

`http://en.wikipedia.org/wiki/In-place_algorithm`

And to learn more about performing merge-sort in-place, read the page

`http://stackoverflow.com/questions/2571049/how-to-sort-in-place-using-the-merge-sort-alg`

[1] In fact, this is not completely true since an intermediate array, or *work array*, `B` is used.

or the page

`http://www.personal.kent.edu/ rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm`

Merge-sort can also be done *out-of-place*, that is, without modifying the input array and by returning a new and sorted array. To see an example, look at the first pseudo-code area at the top of the page

`http://rosettacode.org/wiki/Sorting_algorithms/Merge_sort`

If the in-place solution seems more efficient (as it consumes less resources) the out-of-place is less tricky to implement. For that reason, we want to consider both in this exercise.

1. Write two `Julia` functions for serial merge-sort:
   (a) one in-place version,
   (b) one out-of-place version.
2. Write a parallel `Julia` function based on one of your serial merge-sort functions.
3. It is likely that this parallel function run slower than its serial counterpart. To fix this, one possible trick is to make use of a base-case as we did in the solution of Exercise 3 in Lab 4. You are requested to experiment with different base-cases and collect experimental results.