COSC 6326 Programming Assignment 1

Due: 10am, Thursday, March, 28, 2019 (at the beginning of class).
Submissions will not be accepted afterwards.

In this assignment, you have to design and implement a distributed algorithm for the following fundamental problem: Assume that there are k machines (processors) each having n/k elements (these can be arbitrary positive integers, for example). Thus there are a total of n elements distributed across all the machines. The goal is to find the ith smallest element of all the elements (call this the element of rank i). If rank is n, then we want the largest element among all the elements, and if rank is 1, then we want the smallest element among all the elements. The goal of this assignment is to study this problem in a distributed setting. In particular, we seek efficient algorithms that **communicates as little messages as possible**.

First let's look at the easier problem of finding the element with the largest rank, i.e., the largest element among all the numbers. This can be done easily if all the elements are in one machine (i.e., $k = 1$) — i.e., in a sequential setting. It takes linear time, i.e., by doing $O(n)$ comparisons. (Note that we will only use comparisons to determine, whether an element is larger, smaller or equal to another element). Suppose $k \geq 2$ and each machine has n/k elements. Then there is an easy distributed algorithm: each machine simply finds the maximum element that it has (by running a sequential algorithm locally on its n/k numbers) and sends it to one machine (say the first machine) which simply computes the maximum among all the local maximums received (including the maximum of its own set of n/k elements). This should (hopefully) be faster, since each machine operates in parallel on only n/k elements. Note the total number of elements communicated by this algorithm is only $k - 1$, i.e., each of the $k - 1$ machines sending its local maximum to one machine. On the other hand, a totally naive algorithm would send all of n/k elements to one machine — this requires communicating $(k - 1)(n/k)$ elements which is pretty large.

Now consider the general problem of finding the ith smallest element, $1 \leq i \leq n$, for a given i. Again the goal is to design an efficient distributed algorithm, i.e., one that communicates as few elements as possible. For

example, if $i = \lfloor n/2 \rfloor$, then we are seeking the *median* of all the elements. If all the numbers are in one machine, i.e., $k = 1$, then one can solve the problem in *linear time* (i.e., $O(n)$ time) sequentially either by a deterministic algorithm (see Section 5.2 in [1]) or in expected linear time by a randomized algorithm (see Section 8.6 in [1]). The randomized (sequential) algorithm, in particular, is quite simple, see e.g., Algorithm 27 in [1]. In every iteration a *random* element is chosen as a pivot and the set is partitioned into two sets — all elements less than or equal to the pivot (call it set S_1) and all elements greater than the pivot (call it set S_2). If the number of elements in set S_1 is exactly $i-1$, then the ith smallest element is the pivot. Otherwise if $|S_1| \geq i$, then the ith smallest element is the ith smallest element in S_1. Otherwise, it is the $(i - |S_1| - 1)$th smallest element in S_2 (see Section 5.2 in [1] for correctness proof). This algorithm is linear on average, because, choosing a random element as a pivot on average reduces the size of the problem by a constant factor (see Section 8.6 of [1] for a proof.) Your goal is to implement this algorithm efficiently in a distributed setting where each machine has n/k numbers. (A deterministic algorithm will also work, where the pivot is chosen deterministically.)

Specifically your tasks are:

(1) Write a MPI program where there are k processors each having n/k numbers and the goal is for one machine to output the **largest number** among all the n numbers (see the earlier discussion above). This is a fairly straightforward program in MPI requiring one round of communication (each machine sending its local maximum to one machine).

(2) Do the same as above, but the goal is to output the *median* of all the n numbers. You can use the randomized algorithm mentioned above. Show that this randomized algorithm communicates only $O(k \log n)$ elements on average. (Hint: Show each time the pivot is chosen the number of elements considered reduces by a constant factor.) Describe how to implement this algorithm distributively. Then write an MPI program.

(3) You should demonstrate how your algorithm performs on various inputs on a varying number of machines. You should first test your algorithm on your laptop and see that it works correctly (on relatively small instances), i.e., it correctly finds the median. Most modern laptops have at least 2 to 8 cores (i.e., so many processors), e.g., my Macbook has 4 cores. You should run your program on varying number of machines (for the same input) — say for 1, 2, 4, 8, 16, 32 machines — and study the performance. You should also try different input sizes. You can vary n from say 2^{10}, 2^{11}, $2^{12}, \ldots, 2^{20}$ (at least). One way to generate these numbers is that each machine can generate its own set of (n/k) numbers by randomly sampling from a large-

size set, e.g., from $[0, 2^30]$.

(4) Plot (say, using gnuplot) a graph that shows the the runtime (wall-clock time) of your algorithm as n varies for each fixed $k = 1, 2, 4, 8, ...$

References

[1] G. Pandurangan. Algorithms. `https://sites.google.com/site/gopalpandurangan/home/algorithms-course`