Homework 14

Adam Karl

November 24, 2020

1 Problem 45 (8 points)

1.1 Motivation

Problems (a) and (c) will both be partially dependent on an algorithm we discussed in class that used pairwise comparisons on a CRCW PRAM to find the maximum of n numbers in constant time with n^2 processors.

1.2 a.

To solve this problem in lg(lg(n)) time do:

- let k = the number of remaining values
- while (k>1) do:
  - group the values into k^2 groups, each with n/k values
  - Run the algorithm from class on each group to find the max
  - //since each group has n/k items, each group needs $(n/k)^2$ processors to run in constant time.
  - //since there are k^2/n groups, the total number of processors needed to complete this iteration in constant time is $(n/k)^2 * k^2/n = n$
  - //since we have n processors, this iteration completes in constant time.
- when there is only 1 value remaining, return that value

Since each step takes constant time, our runtime is based on the number of steps that are necessary to solve a sequence of size n.

step (k)	group size (G)	output size (I)
0	1	n
1	2	n/(2^1)
2	4 (=2^2)	n/(2^3)
3	16 (=4^2)	n/(2^7)
4	256 (=16^2)	n/(2^15)

The recurrence relation is:

- G(2) = 2
- $G(k) = (G(k-1))^2$
- I(1) = n
- I(k+1) = I(k)/G(k)

With this in mind, look at the relation between number of steps and the denominator for output size. The algorithm finishes when the output size is 1, so in 1 step we can solve an input of size n=2, in 2 steps we can handle n=8, in 3 steps n=128, in 4 steps n=32768, etc. The pattern is: in k steps, we are able to handle an input of size $n = 2^{(2^k)-1}$. Therefore, (since each step takes constant time) we are able to handle an input of size n in lg(lg(n)) steps (simply take the log of both sides twice). Therefore, our runtime is O(lg(lg(n))).

1.3 b. Max with priority processors

To solve this problem in linear time do:

- Create an array A of length n, each index initialized to false
- Create an answer variable to store the maximum value in the sequence
- Place one processor at each number x_i
- Each processor at x_i should do:
  - concurrently write true to A[x_i]
  - //since each processor is only writing true, it doesn't matter which one writes to A[x_i] if there are multiple values of x_i in the sequence
- Move the processors so that 1 processor is at each index of A, in descending order of priority
  - //the highest priority processor is at index n, the lowest priority processor is at index 1
  - //(the runtime of this algorithm is based on the assumption that this step can be done in constant time)
- Each processor at A[i] should do:
  - if A[i] is true, concurrently write i to the answer variable
  - //due to the prioritization of the processors, only the processor at the maximum true index will write
- return the value stored in the answer variable

1.4 c. Max without priority processors

To solve this problem in linear time do:

- Create an array A of length n, each index initialized to false
- Create an array B of length sqrt(n), each index initialized to false
- Create an answer variable to store the maximum value in the sequence
- Place one processor at each number x_i
- Each processor at x_i should do:
  - concurrently write true to A[x_i]
  - //since each processor is only writing true, it doesn't matter which one writes to A[x_i] if there are multiple values of x_i in the sequence
- //here marks where the algorithm deviates from the solution to (b) in the approach used to find the "right-most" true in A
- Place one processor at each index of A
- Each processor at A[i] should do:
  - if A[i] is true, concurrently write true to B[$\lfloor$i/sqrt(n)$\rfloor$]
  - //the idea is that processors at A[1] through A[sqrt(n)] all point to B[1]
  - //A[sqrt(n) + 1] through A[2*sqrt(n)] all point to B[2]
  - ...
  - //A[n-sqrt(n)] through A[n] all point to B[sqrt(n)]
- Run the algorithm from class to find the max of sqrt(n) numbers in constant time with n processors. The input is each INDEX i of B for which B[i] is true.

- //the algorithm returns which sqrt(n)-size "group" of A has the right-most true

- take the solution to the algorithm (say: z) and go the the z-th group in A.
- Run the algorithm from class again on the z-th group of A (with all n processors) to find the right-most index of A for which A[i] is true. Return this index i as the answer.