Homework 25

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

31 October 2016

Please provide written or oral feedback on this and future homeworks :)

Problem 17:

Although a CRCW Common algorithm for finding the max of n numbers was developed earlier, it required n^2 processors and could be shown to be highly inefficient. An improvement to the parallel efficiency of this problem can be made at the sacrifice of runtime if only n processors are used. Unlike the $\mathcal{O}(1)$ CRCW algorithm originally developed, this algorithm is modeled after the EREW divide and conquer algorithm.

In the EREW algorithm, recursive calls were made to divide the initial set of numbers into halves with each half given half the processors. This process continues until each processor is assigned a single number at which point two processors assigned neighboring numbers make a comparison to determine the larger. The larger value is then returned from the recursive call.

In a similar manner, for this CRCW common algorithm recursive calls can be made until each processor is assigned a single number. However, instead of each processor only comparing with the neighboring processor, it can communicate with both of neighbors (Note: there is only one neighbor for processors assigned numbers at the end). This CR done by the three neighboring processors can be used to determine which of the processors holds the largest

number. Since the rank of the processor holding the largest number will be a common value, it can be safely written to a single location which can then be returned from the recursive call. In the case that two processors are assigned the same number, an arbitrary decision can be made, since in this case, the commonality will be the number written to the shared location.

As the process returns from its recursive call tree, the number of available numbers to be assigned processors decreases; however, instead of having the processors unassigned do no work, the can be used to further reduce the problem size by expanding the size of the local group for which a max is found by double assigning a number to two or more processors.

As a result of cutting the size of the problem into progressively smaller groups as the recursive tree returns, this algorithm will have a run time of $\log \log n$.

Problem 18:

This was a fun problem.

First, initialize a bitarray A of length n, will all bits zeroed. The ith bit from the right represents whether or not the value i has been found in the input sequence $x_1 \dots x_n$. Now, assign each of the n processes a unique value i from 1 to n. Have the ith processor look at x_i. Let that value be k. Flip the kth bit from the right in the bit array A. This operation occurs in constant time, since there are n indexes into the sequence $x_1 \dots x_n$, and each operation takes constant time. Notice that if a value occurs twice in the input sequence, then the same index in A will be accessed by multiple processors at once, but they will be writing a common value.

Finally, assuming each processor is assigned a unique priority from 1 to n as per the problem description, have the processor with priority y look at the bit $A[n - y + 1]$, and if it's set, it will write $n - y + 1$ to the output register. (That is, assign the highest priority processor to the highest index in the bit array, and write that index to output if the value at that index is set). Because the highest priority processor takes precedence to all lower priority processes, the highest value will be written to the output register.

Problem 19:

The beginning steps of Problem 19 are the same as problem 18. I will repeat them here for the sake of completeness, but feel free to skip to the final paragraph if you understand what I'm attempting to communicate.

First, initialize a bitarray A of length n, will all bits zeroed. The ith bit from the right represents whether or not the value i has been found in the input sequence $x_1 \dots x_n$. Now, assign each of the n processes a unique value i from 1 to n. Have the ith processor look at x_i. Let that value be k. Flip the kth bit from the right in the bit array A. This operation occurs in constant time, since there are n indexes into the sequence $x_1 \dots x_n$, and each operation takes constant time. Notice that if a value occurs twice in the input sequence, then the same index in A will be accessed by multiple processors at once, but they will be writing a common value.

Finally, we need to find a way in constant time to write out the highest bit in the array. This is actually fairly easy; cast the bit array as an integer (making the unrealistic assumption that we can perform computation with 2^n bits, like we have with past problems). Let that value be B. Write to the output register the value $\lfloor lg(B) \rfloor$. This is, of course, constant time arithmetic.