Homework 23

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

25 October 2016

Problem 9:

In order to compute a prefix that is also a suffix, we can look to the Knuth-Morris-Pratt string pattern matching algorithm. In KMP, the first step is to build a table out of the pattern we are searching for. In this table at each letter in the pattern a number is stored which is equal to the longest prefix that is also a suffix at that point in the pattern.

This is demonstrated in the table below:

Table 1: KMP Table

	0	1	2	3	4	5	6	7
Pattern	a	b	a	b	a	b	c	a
Value	0	0	1	2	3	4	0	1

If the length of the pattern is n, then the time required to generate this table sequentially is also $\mathcal{O}(n)$. Unfortunately, on a EREW system, this table generation cannot be parallelized because of the data dependencies in that previous table index values are sometimes needed in order to compute the current table index value, i.e. table[i] = table[i-1].

To go through this table, we can use the same method of finding the maximum number given a sequence of integers that we did in class.
The code for this is below:

Function: $Max(intx_1 \cdots x_n, p)$ `/* p = number of processors */`
return $max(Max(x_1 \cdots x_{n/2}, p/2), Max(x_{(n/2)+1} \cdots x_n, p/2))$

This algorithm takes $O(n/p + \log n)$ time to run on p processors.

Therefore, these two algorithms combined with p = n processors will run in $\mathcal{O}\big(n \log n\big)$ time.

Problem 10:

This problem is basically the same as the previous problem, except that now we are on a CRCW system. This means that now we can effectively parallelize the KMP table generation. This can be done by assigning a processor to a certain chunk, n/p, of the table generation algorithm in which each processor can compute the value of an index in the table. Which if the number of processors p $\geq$ n, then the KMP table generation algorithm will run in $\mathcal{O}\big(1\big)$ time.

Then we can use the CRCW max algorithm that was done in class in order to parallelize finding the max value in the KMP table. The algorithm generates a 2D array T, where each index is either a 0 or 1. If a 1 is placed at index T[i, j], then $x_i < x_j$, else a 0 is placed at T[i,j]. This is then repeated for each i and j. In order to find which is the max value, we can just OR each row in the table, and which ever row is filled only with 0's is the max value.

The algorithm is as follows:

$Max(x_1 \cdots x_n, p)$ `/* p = number of processors */`

if $x_i < x_j$ **then**
  $T[i,j] = 1$
else
  $T[i,j] = 0$

Finding the max value from this table T:

$FindMax(T[][], p)$ `/* p = number of processors */`

$A[i] = 0$ `/* Create a new array A for each row */`

if $T[i,j] = 1$ **then**
  $A[i] = 1$
if $A[i] = 0$ **then**
  return x_i

Because the table T has n^2 elements, p = n^2 processors would allow the Max function to run in $\mathcal{O}(1)$, or constant time by assigning a processor to each index in the table. The $FindMax$ function also has n^2 elements to go through, again requiring p = n^2 processors in order to reach $\mathcal{O}(1)$ time.

Therefore, running the KMP table generation, Max, and $FindMax$ with p = n^2 processors will achieve $\mathcal{O}(1)$time.

Problem 11:

a,b:

Sadly, I was unable to make any progress on this problem. For problem **a**, I could not determine what to do with the carry bit that might occur on addition. Divide and conquer in either continuous halves or alternating halves did not appear to afford an apparent solution as to where to place carry bits; however, i do believe they would be the **CR** portion of this **CREW** algorithm.

For **b**, I believe the strategy would be to have the the processors continue

keep track of the carry bits as they move up the tree. This is possible because there are now n^2 processors available; however, it comes at the cost that at every node in the recursive tree $\log n$ work will have to be done. Since the tree is $\log n$ in depth, the total run time of this algorithm will be $\log^2 n$.