CS1510 Parallel Problems 9, 10, & 11

Rebecca Negley, Sean Myers

November 9, 2011

9. The input to this problem is a character string C of n letters. The problem is to find the largest k such that
   C[1]C[2] ... C[k] = C[n - k + 1] ...C[n 1]C[n]
   That is, k is the length of the longest prefix that is also a suffix. Give a EREW parallel algorithm that runs in poly-logarithmic time with a polynomial number of processors.

   Solution: The first thing to do is to kind of make this a pseudo-CRCW problem. We are going to take the array C, and copy it n times. This will take log(n) time, because at if you need to copy it n times, but can only use one processor each time, then you just copy C into C_2 first. Then you have two arrays which you can read from, and write to two new arrays! Hence, log(n) if we have enough processors to copy. In order to have enough processors, each processor will have 1 index into the array, and since we need n copies, and the array is size n, we need n^2 processors.

   Now that we have n copies, and n processors per copy, we can now find k via brute force. We pass k into each processor group, with the array copy C[k][n]. We then create a local array that is the size of k, to put a boolean value into. Now we check each value to see if(c[k][k]==C[k][n-k+1]) for all possible k iterations. This operation takes O(1) for all sets, since it is all done in parallel, and if it is true, put a 1 into the array, otherwise 0. They aren't concurrent writes, since no two processors write to the same memory location.

   Now that you have n boolean arrays of size k, where k goes from 1 to n, we do an AND on all of these and insert it into a new array of size n. This operation takes time log(n) in parallel. Now with THIS array, we now have the truth value of all k. Now we use MAX on the indeces of this array (If you don't know the index from the start, then you can make an array of 1's and subset sum to find it in log(n) time), where it will return the max k value that has a 1. This is the answer.

   PSEUDO-ALGORITHM/CODE:
   ```
   Create $n$ arrays of size $n$, kPossible[n][n]
   Pass $n$ processors to each row, with a $k$ value spanning from 1 to n
   Create a new boolean array of size k kAND[k] (there will be n of these)
   Compare in parallel kPossible[k][k]==kPossible[k][n-k+1] and store that into kAnd[k]
   Create a global array kMax[n].
   AND all the values of each group of k, and store each of those in kMax[n]
   find the max indeces of kMax where kMax[i] = 1
   return index.
   ```

   The total run time is roughly 3log(n) + 3. Which reduces to O(log n). (Side note, if you can could you direct me of a possible tree breakdown that might make this program more elegant, I couldn't find one :( )

10. The input to this problem is a character string C of n letters. The problem is to find the largest k such that C[1]C[2] ... C[k] = C[n - k + 1] ...C[n - 1]C[n] That is, k is the length of the longest prefix that is also a suffix. Give a CRCW parallel algorithm that runs in constant time with a polynomial number of processors.

Solution: If we have n^2 processors, then we can solve this problem in O(1) time. To do this, we delegate n processors to each k, where k can be 1 to n-1.

We break this into three steps: Since each processor has a k and an index into the array to check, check if array[i] == array[n-i+1]. If it does, store a local 1 for the second step. The second step is do an AND on all the bits for each k group delegated. There will now be an indexed array of k groups that will either have a 1 or a 0. Lastly, get the max index of the array that has a 1 in it. Return that value.

Example: An array of [A, B, A]. there will be 2 k groups of processors, and each group will have 3 processors. The first group will check only if(A == A) which is true. The second group will check if AB == BA, which is false So the final k array will be [1,0]. We take a max of the indeces into that array, and get that the max index with a 1 in it is 1, hence $k = 1$.

11.

12. Design a parallel algorithm for adding two n bit integers. Your algorithm should run in $O(log\ n)$ time on a CREW PRAM with n processors.

Solution: We will use a divide a conquer approach to break down this problem. We are given two n-bit integers $A = a_n a_{n-1} \dots a_1$ and $B = b_n b_{n-1} \dots b_1$. We can use $n/2$ processors to add $a_n \dots a_{n/2+1}$ with $b_n \dots b_{n/2+1}$ and the other $n/2$ processors to add $a_{n/2} \dots a_1$ with $b_{n/2} \dots b_1$. With this information, we pass up the carry bit and the rightmost 0 of each. Then, if the carry bit on the lower order addition is 1, we use at most $n/2$ processors to flip all bits (right to left) up to and including the rightmost zero in the higher order sum. If there is no 0 in the upper level sum (rightmost zero defined to be infinity), flip all of them and mark the carry bit from 0 to 1. We cannot have a carry bit of 1 without a 0 in the string. (This is embarrassingly parallel, so we can do this in constant time.) We then let $A + B = C$ be the n-bit number with higher order bits from the higher order sum and lower order bits from the lower order sum. To find the rightmost 0 of C, we take the rightmost 0 of its lower order terms. If the rightmost 0 is defined to be infinte in the lower order terms, we take the rightmost 0 of the upper level terms ($+n/2$ to account for new position in C). We do not need to worry about it being flipped because, if there are no 0s in the lower order terms, there can be no carry bit from the lower order terms. We then define the carry bit of C to be the carry bit of the upper order terms. We can then pass this information up to the next level. When we get to the topmost level, we can add the carry bit by appending it to the highest order term. The time of this algorithm is $T(n, n) = T(n/2, n/2) + 1 = O(log\ n)$. Its efficiency is $E(n, n) = \frac{n}{n*log\ n} = \frac{1}{log\ n}$. In code:

```
add($a_n \dots a_1$,$b_n \dots b_1$,$p = n$)
    ($c_n \dots c_{n/2+1}$,RMZ2,CB2)=add($a_n \dots a_{n/2+1}$,$b_n \dots b_{n/2+1}$,$p = n/2$)
    ($c_{n/2} \dots c_1$,RMZ1,CB1)=add($a_{n/2} \dots a_1$,$b_{n/2} \dots b_1$,$p = n/2$)
    if CB1=1 do:
        parallelfor $i$ from $n/2 + 1$ to min(RMZ2,$n/2$) do:
            flip $c_{i+n/2}$
        if RMZ2= $\infty$ do:
            CB2=1
    RMZ=RMZ1
    if RMZ= $\infty$ do:
        RMZ=RMZ2+$n/2$
    return ($c_n, \dots, c_1$,RMZ,CB2)
```