CS1510 Parallel Problems 15,17,18,20

Rebecca Negley, Sean Myers

November 14, 2011

16. Give an algorithm for the minimum edit distance problem that runs in poly-log time on a CREW PRAM with a polynomial number of processors. Here poly-log means ($log^k n$) where n is the input size and k is some constant independent of the input size.
Recall that the input to this problem is a pair of strings $A = a_1...a_m$ and $B = b_1...b_n$. The goal is to convert A into B as cheaply as possible. The rules are as follows. For a cost of 3 you can delete any letter. For a cost of 4 you can insert a letter in any position. For a cost of 5 you can replace any letter by any other letter
Solution: The first step to creating this algorithm is looking at the algorithm from the past dynamic programming algorithm:

```
Initdist[m,n]
for i = 0 to m
    dist[i,0] = i
for j = 0 to n
    dist[0,j] = j

for i = 1 to m
    for j = 1 to n
        if(A[i] == B[j]): d[i,j] = d[i-1, j-1]
        else: d[i,j] = min((d[i-1,j] +3, d[i,j-1]+4, d[i-1,j-1]+5)
return d[m,n]
```

This algorithm can be easily parallelized with n^2 processors. We just need to run all i and j in parallel. We learned in class that min on an EW machine is going to take log(n) time.

To initialize the array, it will take O(1) time, since both loops can be parallelized.

The branch conditional for all i and j will take O(1) time. The main problem is the min on the else branch. In the worst case, all processors hit this conditional, in which case all processors in parallel will be looking at O(1) + O(logn) run time, as the total runtime (the first O(1) is from the if calculation). None are writing concurrently to the same spot since all i and j are in parallel.

17. Design a parallel algorithm that finds the maximum number in a sequence $x_1...x_n$ of (not necessarily distinct) integers. Your algorithm should run in time $O(\log \log n)$ on a CRCW PRAM with n processors.

Solution: Observe that, in $O(1)$ time, we can find the maximum of k integers with k^2 processors using the CRCW algorithm described in section (4) of the parallel notes. (The notes find the minimum, but checking if $x[i] \geq x[j]$ instead of $x[i] \leq x[j]$ converts this to a maximum algorithm.) We will use this fact below. First, to compute the maximum of n values with n processors, we split up our n values into $\sqrt{n}$ sets of $\sqrt{n}$ numbers and assign each set $\sqrt{n}$ processors. When the maximum

of these sets is known, we must compute the maximum of the $\sqrt{n}$ returned values with n processors, which can be done in time $O(1)$ as noted above. Hence, the parallel recurrence relation for the time of this algorithm is $\mathrm{T}(n, n) = \mathrm{T}(\sqrt{n}, \sqrt{n}) + 1$. As we have seen before, the total time is then equal to the depth of recursion tree. How deep is this recursion tree? We stop when each set has only 2 values because one processor can compute the maximum of 2 values in one timestep. The degree of the sets is cut in half at every step. Then, at the deepest level of the tree:

$$\begin{aligned} n^{1/2^d} &= 2 \\ \frac{1}{2^d} \log n &= 1 \\ \log n &= 2^d \\ \log \log n &= d \end{aligned}$$

Hence, since each level takes time 1, the algorithm runs in time $O(\log \log n)$.

18. Design a parallel algorithm that finds the maximum number in a sequence $x_1 ... x_n$ of (not necessarily distinct) integers in the range 1 to n. Your algorithm should run in constant time on a CRCW Priority PRAM with n processors. Note that it is important here that the x_i's have restricted range. In a CRCW priority PRAM, each processor has a unique positive integer identifier, and in the case of write conflicts, the value written is the value that the processor with the lowest identifier is trying to write.

    Solution: Create an n-sized array A, initially zeroed out. (Have each processor assign a 0 to a unique location to start.) Assign each processor to one of the input numbers. Each processor reads its input number $i \in [1, n]$ and writes a 1 to $A[i]$. Then, assign processor 1 to $A[n]$, processor 2 to $A[n-1]$,..., and processor n to $A[1]$. Each processor reads its location in the array, and if it is a 1, it writes its array location to the return value. The value written by the highest priority processor will be the highest array location containing a 1. Since an array location contains a 1 if and only if there is an input number with that value, our algorithm must be correct. Since each of the three steps takes constant time, the algorithm must run in time $O(1)$.

20. Show that if there is an an algorithm for a particular problem that runs in time $T(n, p)$ on a p processor CRCW machine, then there is an algorithm for this problem that runs in time $T(n, p)$ log p on a p processor EREW machine.
    Solution: In order to break apart a concurrent read, we need to duplicate an array.

    Let's say that at any point, if there is contention over a single read memory block, let's call it memory location k from two groups of processors p_1 and p_2, we copy over the memory location k into two memory locations k_1 and k_2. The copying will take O(1) time.

    Localizing this to n processors is recursively creating two groups $group_{p1}$ and $group_{p2}$. Do the copy like above. Now let's say $group_{p1}$ has 2 processors in it (same for $group_{p2}$), then we do another recursive copy using k_1 for $group_{p1}$ and k_2 for $group_{p2}$. The algorithm for resource exclusion would look something like this:

```
resourceAllocation(groupOfProcessors g, data k):
    if(g.size == 1): return;
    else:
        group1 = [lowestSizeInGroup,g.size/2]
        group2 = [g.size+1/2, highestSizeInGroup]
        data k2 = k
        parallel resourceAllocation(group1, k)
```

parallel resourceAllocation(group2, k2)

This will make it so that all are exclusive reads, because any concurrent data being read has now been made exclusive to an individual processor. This runs in log(p) time, since we divide p each time and run it in parallel.

We also have to worry about concurrent writes, but that is almost identical to concurrent reads. The one difference though is that there needs to be some condition to what to write to the buffer. For example, let's say there is a concurrent write of two processors p_1 and p_2. For a min function, whichever processor holds the minimum value, puts in their value. If it is an or, then any processor with a 1 puts in its value. So let's say there is some boolean comparison comp(p1, p2), where if it is true, p1 writes, otherwise p2 writes. This way, it is problem independent as long as comp(p1, p2) is changed with the problem input.

The algorithm would look something like this:

```
Answer(processors p):
    if(p.groupsize == 1): return answer[p] ;
    else:
        ans1 = parallel Answer([lowestSize, p/2])
        ans2 = parallel Answer([p+1/2, highestSize])
        if(comp(ans1,ans2)): return ans1
        else: return ans2
```

Again, we see that there is parallelization to the point where the time will only be log(p) to get the correct answer written.

Therein, we have now turned a CRCW algorithm that runs in T(n,p) time to an EREW algorithm in T(n,p)*log(p)