CS1510 Parallel Problems 19, 21, & 23

Rebecca Negley, Sean Myers

November 16, 2011

19. 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 Common PRAM with n processors.

    Solution: As in problem 18, create an array A of size n, initially 0. Have each processor read one value x_i and write 1 to $A[x_i - 1]$. Now, we want to determine the greatest array index where a 1 is written. First, split the array into $\sqrt{n}$ groups of size $\sqrt{n}$ (so $A[0], \ldots, A[\sqrt{n} - 1]$ is the first group, $A[\sqrt{n}], \ldots A[2\sqrt{n} - 1]$ the second, etc). Using $\sqrt{n}$ processors for each of these groups, we can OR the all of the values within each group in one timestep. Create a $\sqrt{n}$-sized array B. Fill B such that $B[i]$ =OR$(A[i*\sqrt{n}], \ldots, A[(i+1)*\sqrt{n}-1])$ from the results of the step above. Then, we want to find the maximum index i in B such that $B[i] = 1$. We have already seen that we can take the maximum of $\sqrt{n}$ values with n processors in constant time. Given that j is the maximum index in B such that $B[j] = 1$, we know the maximum index with a corresponding 1 in A must be in $A[j * \sqrt{n}], \ldots, A[(j+1) * \sqrt{n} - 1]$. There are at most $\sqrt{n}$ values in this set, and we again have n processors. Hence, we can find and return this value in constant time, so the whole algorithm will run in constant time. In pseudocode:

    MAX($x_1, \ldots, x_n$)
    par for i from 0 to $n - 1$ do:
    $A[i] = 0$
    $A1[i] = 0$
    par for i from 1 to n do:
    $A[x_i - 1] = 1$
    $A1[x_i - 1] = x_i$ //what we want to take max over
    par for i from 0 to $\sqrt{n} - 1$ do:
    $B[i]$ =OR$(A[i * \sqrt{n}], \ldots, A[(i + 1) * \sqrt{n} - 1])$
    $B1[i] = B[i] * i$ //what we want to take max over: i if OR=1, 0 if OR=0
    j =MAX$(B1[0], \ldots, B1[\sqrt{n} - 1])$
    return MAX$(A1[j * \sqrt{n}], \ldots A1[(j + 1) * \sqrt{n} - 1]$

21. Give a parallel algorithm for the following problem that runs in time $O(\log n)$ on an EREW PRAM. The input is a binary tree with n nodes. Assume that each processor has a pointer to a unique node in the tree. The problem is to number the leafs consecutively from left to right (that is in-order). Note that this algorithm is needed for the algorithm in the nodes for computing arithmetic expressions.

    Solution: To do this, we need to reduce the problem to pointer doubling to solve for indexing (which we know is $\log(n)$). The trick will be to make it in such a way that the node pointing to, say index 7 when we are done with the linked list, will be the correct corresponding label for the original node that the processor was assigned to.

    The algorithm to create such a linked list is as follows:

```
LinkedDeque createLink(root):
    LinkedDeque left, right;
    if(root.left != NULL) parallel left =createLink(root.left)
    if(root.right != NULL) parallel right=createLink(root.right)
    LinkedDeque ret;
    ret.addBack(Left)
    ret.addBack(new Node(root, 1))
    ret.addBack(Right)
    return ret;
```

If we call this algorithm from the top of a binary search tree that holds n nodes, we can see that since the left and right subtrees create in parallel, and the linking of the linked lists takes constant time at each node, it will take $\log(n)$ time to build this tree. The tree itself will hold the correct corresponding Node so that when the pointer doubling is done, the processors can update themselves in parallel (taking O(1) time).

Overall, since we know pointer doubling takes log(n) time, this eularian tour takes log(n) time, and the updating of the nodes once the pointer doubling is done takes O(1) time, we are looking at an algorithm that takes: log(n) + log(n) + 1 time, which is equivalent to O($\log(n)$) time.

23. Design a parallel algorithm that takes a binary expression tree, where the leaves are integers, and the interval nodes are the four standard arithmetic operators addition, subtraction, multiplication, and division, and computes the value of the expression. Your algorithm should run in $O(\log\ n)$ time on a CREW PRAM with n processors, where n is the number of nodes in the tree. You may assume that each processor initially has a pointer to a unique node in the tree.

Soluion: First we will prove that the collection C of functions $\{f(x) = \frac{ax+b}{cx+d}\}$ is closed under addition, subtraction, multiplication, and division with constants and under composition. Let $f_1(x) = \frac{a_1x+b_1}{c_1x+d_1}$, $f_2(x) = \frac{a_2x+b_2}{c_2x+d_2}$, and k be some constant. Then,

$$\begin{aligned}(f_1(x)+k) = (k+f_1(x)) &= \frac{a_1x+b_1}{c_1x+d_1} + k \\ &= \frac{a_1x+b_1+k*c_1x+k*d_1}{c_1x+d_1} \\ &= \frac{(a_1+k*c_1)x+(b_1+k*d_1)}{c_1x+d_1} \in C.\end{aligned}$$

Also,

$$\begin{aligned}(f_1(x)*k) = (k*f_1(x)) &= \frac{a_1x+b_1}{c_1x+d_1} * k \\ &= \frac{(a_1*k)x+(b_1*k)}{c_1x+d_1} \in C.\end{aligned}$$

Since subtraction and division not associative, we must consider $(f_1(x)-k)$ separately from $(k-f_1(x))$

and $(f_1(x)/k)$ separately from $(k/f_1(x))$. Observe that

$$\begin{aligned}(f_1(x)-k) &= (\frac{a_1x+b_1}{c_1x+d_1}-k)\\ &= \frac{a_1x+b_1-k(c_1x+d_1)}{c_1x+d_1}\\ &= \frac{(a_1-k*c_1)x+(b_1-k*d_1)}{c_1x+d_1} \in C,\end{aligned}$$

and

$$\begin{aligned}(k-f_1(x)) &= k-\frac{a_1x+b_1}{c_1x+d_1}\\ &= \frac{k(c_1x+d_1)-a_1x-b_1}{c_1x+d_1}\\ &= \frac{(k*c_1-a_1)x+(k*d_1-b_1)}{c_1x+d_1} \in C.\end{aligned}$$

We also have

$$\begin{aligned}(f_1(x)/k) &= \frac{a_1x+b_1}{c_1x+d_1}/k\\ &= \frac{a_1x+b_1}{(k*c_1)x+(k*d_1)} \in C,\end{aligned}$$

and

$$\begin{aligned}(k/f_1(x)) &= k/\frac{a_1x+b_1}{c_1x+d_1}\\ &= k*\frac{c_1x+d_1}{a_1x+b_1}\\ &= \frac{(k*c_1)x+(k*d_1)}{a_1x+b_1} \in C.\end{aligned}$$

It then only remains to show that $f_1(f_2(x)) \in C$.

$$\begin{aligned}(f_1(f_2(x))) &= f_1(\frac{a_2x+b_2}{c_2x+d_2})\\ &= \frac{a_1(\frac{a_2x+b_2}{c_2x+d_2})+b_1}{c_1((\frac{a_2x+b_2}{c_2x+d_2})+d_1}\\ &= \frac{\frac{(a_1*a_2)x+(a_1*b_2)+b_1(c_2x+d_2)}{c_2x+d_2}}{\frac{(c_1*a_2)x+(c_1*b_2)+d_1(c_2x+d_2)}{c_2x+d_2}}\\ &= \frac{(a_1*a_2)x+(a_1*b_2)+b_1(c_2x+d_2)}{(c_1*a_2)x+(c_1*b_2)+d_1(c_2x+d_2)}\\ &= \frac{(a_1*a_2+b_1*c_2)x+(a_1*b_2+b_1*d_2)}{(c_1*a_2+d_1*c_2)x+(c_1*b_1+d_1*d_2)} \in C.\end{aligned}$$

We can evaluate functions in C in constant time. Since each operation takes constant time, we can define our cuts using the above operations and use the same algorithm described in section (11) of the parallel notes.