Homework VI Group problem 16

Graded by: Group 7 (Susan Joseph, Su Qi, Bradford Hovinen)
Adopted from a solution by: Group 7 (Susan Joseph, Su Qi, Bradford Hovinen)

Problem Statement

Suppose we are given a set of n elements stored in an array A together with an array L such that L_i is a label for A_i. Each label L_i is an integer in the range 1 to k, where k is a constant. Develop an $O(\lg n)$ time, $O(n)$ work EREW algorithm that rearranges the elements of A such that A consists of all of the elements with label 1, followed by all of the elements of label 2, followed by all of the elements of label 3, and so on. Further, all of the elements of label 1 should be in the same order in the rearranged A as they were in the original A.

Solution

The algorithm defined below works by finding the maximum label present in L through a randomized parallel prefix algorithm and acting on all the nodes that have that label. It then eliminates that label from consideration and finds the next highest label, continuing until all array elements have been processed. The variable l keeps track of the current label. The procedure Randomized-List-Prefix, invoked at two different points in the algorithm, is defined in CLR section 30.4 and shall not be repeated here. The three arguments to that procedure are the input array, the procedure to invoke over the array, and the output array, which can be nil if not needed or the same as the input array if desired. The procedure Max returns the maximum of the two inputs, while the procedure Sum returns the sum of the two inputs. These two procedures are passed as arguments to Randomized-List-Prefix; the return value is assumed to be the field $d[A_n]$, where n is the length of the array A supplied to that procedure.

proc ARRANGE-A(A, L) $\equiv$

var $C_k, 1 \leq k \leq \lfloor \frac{length[A]}{\lg length[A]} \rfloor$: array of numbers

$D_k, 1 \leq k \leq length[A]$: array of numbers

$E_k, 1 \leq k \leq \lfloor \frac{length[A]}{\lg length[A]} \rfloor + 1$: array of numbers

l, b, i, j : number;

$l \leftarrow 1$

$b \leftarrow n$

Assign corresponding elements of A and L to processors $P_1, \ldots, P_m$ $(m = \frac{n}{\lg n})$ such that each processor gets an adjacent sequence of $\lg n$ elements. Let E_i be the index in A of the first element managed by P_i and let $E_{m+1} = n + 1$;

while $l \geq 1$

do $l \leftarrow$ RANDOMIZED-LIST-PREFIX$(L, \text{MAX}, \text{NIL})$

if $l < 1$ **then** break

for each processor P_i in parallel **do**

$C_i \leftarrow 0$

for $j := E_i$ **to** $E_{i+1} - 1$ **do**

if $L_j = l$

then $C_i \leftarrow C_i + 1$

$L_j \leftarrow -1$

$b \leftarrow b -$ RANDOMIZED-LIST-PREFIX(C, SUM, C)

for each processor P_i in parallel **do**

for $j := E_{i+1} - 1$ downto E_i **do**

if $L_j = l$

then $D_{b+C_i} \leftarrow A_j$

$C_i \leftarrow C_i - 1$

for each processor P_i in parallel **do**

for $j := E_i$ **to** $E_{i+1} - 1$ **do**

$A_j \leftarrow D_j$

end

The assignment on line 7 forces the while loop to run through at least one iteration, while the assignment on line 8 initializes the index in the array C to which values will be written during rearrangement. Lines 10 through 12 assign the elements of A and L to the processors. It is assumed according to the arithmetic model of computation that this can be done in $O(1)$ for each processor, with a total work of $O(\frac{n}{\lg n})$.

The call to RANDOMIZED-LIST-PREFIX on line 15 returns the maximum of all the labels in the array L. Using the implementation given in CLR section 30.4, it will achieve time $O(\lg n)$ and work $O(n \lg n)$ with extremely high probability. The test on line 16 gives termination conditions for the loop. If the label returned by the aforementioned call to RANDOMIZED-LIST-PREFIX is not valid, then all the elements with valid labels have been processed, so the loop terminates.

The for loop on lines 18-23 then initializes the array C, which has one element per processor, with the number of nodes in the block managed by that processor that have the maximum label, given in the variable l. In addition, as the processor traverses the nodes it has been assigned, it replaces with -1 the entries in L corresponding with the maximum labels, so they will not be considered in future iterations. This requires that each processor scan through no more than $\lceil \lg n \rceil$ nodes, and therefore requires time $O(\lg n)$ and work $O(n)$.

Once C is initialized, another call to RANDOMIZED-LIST-PREFIX is made on line 27. This call replaces the value of each element i in C with the sum of the values of all elements $j : 1 \leq j \leq i$. It also returns the total sum of all elements in C. Hence, the return value is the total number of elements in A that have labels equal to the current maximum label l. This, in conjunction with the values in C, is used to place the elements of A in their rearranged order in the array D. The call to RANDOMIZED-LIST-PREFIX operates on $\frac{n}{\lg n}$ nodes and therefore runs in time $O(\lg \frac{n}{\lg n}) = O(\lg n)$ and requires work $O(\frac{n}{\lg n}) = O(n)$.

Once modified on line 27, the variable b contains the index in D immediately preceding the one into which the element with the lowest index that has the maximum label l should be written. The elements in C, when added to b, contain the indices into which the elements of A with the highest index and the maximum label for each block should be written. Hence the formula for determining the index in D into which the highest indexed node in a given block i with the maximum label l should be written is $b + C_i$. In this manner, the original order in A can be kept. Diminishing

C_i by one yields the index for the node with maximum label with the next highest node. The for loop on lines 29 through 33 does just that. Each processor scans through its block backwards, starting with the highest indexed node. If a given node has the maximum label, it is placed at the aforementioned index in D, and the value of C_i is diminished by one to reflect the index for the next node of maximum label to be processed. As with the for loop on lines 18-23, this loop performs an $O(1)$ operation on each node managed by the processor, and therefore requires time $O(\lg n)$ and work $O(n)$.

On the next iteration, all the labels that had the maximum value l are replaced with -1, which becomes maximum only after all the nodes have been processed. Hence, the next call to RANDOMIZED-LIST-PREFIX will return the next-highest label and processing will continue on the nodes with that label. When the highest label is -1, all the nodes have been processed, so the test on line 16 breaks out of the loop.

In the final for loop on lines 39-41, each processor iterates through its block of nodes, copying it back from the sorted array D to the input array A, as is implied in the problem statement. It is of the same form as the loops described above and therefore requires the same time and work.

That this algorithm is EW is clear. In every parallel step, that is, in every for loop, each processor writes values either to its specific index in C or to its assigned block of indices in the arrays D and A. RANDOMIZED-LIST-PREFIX is also EREW according to CLR section 30.4. The two global variables, b and l, are not modified at all in the parallel portions of this algorithm, so they do not affect at all whether this algorithm is EW.

There is one small obstacle in the algorithm being ER, however. There are two global variables that must be accessed by all processors simultaneously: b and l. The act of distributing a copy of these values to each processor is not explicitly stated in the algorithm for the purpose of clarity. It can, however, be done in time $O(\lg m)$ and work $O(m \lg m)$, where $m = \frac{n}{\lg n}$ is the number of processors. Under this scheme, one processor first makes a copy of the global variable, leading to two copies. Two processors can then each make one copy, leading to four copies. The process continues until every processor has a copy. This scheme is clearly EREW. Since every stage doubles the number of copies of the variable in $O(1)$ time, the scheme clearly requires time $O(\lg m)$ and by definition cannot require more than $O(m \lg m)$ work. Therefore the global variables can be distributed to all processors without disturbing the asymptotic order of either the time required or the work done, as long as it is done only a constant number of times.

Since the labels are positive integers from 1 to some constant k, inclusive, the total number of distinct labels cannot exceed k. Therefore, the number of iterations through the while loop on lines 14-33, which runs one iteration for each distinct label, cannot exceed k. Each iteration requires time $O(\lg n)$ and work $O(n)$ as described above. Since k is a constant, it has no effect on the asymptotic order of the algorithm, and hence the algorithm requires time $O(\lg n)$ and work $O(n)$.