Homework 32

Brian Knotten, Brett Schreiber, Brian Falkenstein

November 12, 2018

18

The outline for this algorithm is as follows: Since the input is a list of (possibly overlapping) integers $x_1...x_n$ between 1 and n, then each processor p_i of the n processors can look at its corresponding integer x_i and confirm that the number x_i is in the input by writing a 1 to an (originally 0) memory location $IsInInput[i]$. When $IsInInput[i] = 1$, the integer i is in the input. If no processor writes to $IsInInput[i]$, then it stays 0, which means that the integer i is not in the input. This stage of the algorithm takes only constant time, since n processors working on n integers each do one read step and one write step.

Next, each processor p_i will read $IsInInput[n - i + 1]$. If p_i finds a 1, then it will write $n - i + 1$ to the memory location $Maximum$. $Maximum$ will contain the maximum of $x_1...x_n$ after this step, since the lowest numbered processor will write the highest valued number. If p_i reads a 1 from $IsInInput[n - i + 1]$, then it will write $n - i + 1$ to $Maximum$ which will be the maximum if and only if all $p_j, j < i$ read a 0 in $IsInInput[n - j + 1]$, which implies there are no higher numbers in the input. If there is a larger number, then a lower register (with a higher priority) will write to $Maximum$ instead. This step also takes constant time, because each processor performs one read and one write.

Algorithm 1 CRCW Priority $O(1)$ algorithm for maximum with n processors.

Require: Input $x_1...x_n$, an n-sized memory location $IsInInput$, and a memory location $Maximum$.

```
IsInInput[i] ← 0                          ▷ First, zero out the IsInInput array.
IsInInput[x_i] ← 1   ▷ This is a CRCW Common step, since all processors write the same number: 1.
if IsInInput[n − i + 1] == 1 then
    Maximum ← n − i + 1                   ▷ This is the CRCW Priority step.
end if
if i == 0 then                            ▷ Designate one processor
    Output Maximum                        ▷ to output the maximum.
end if
```

19

The outline for this algorithm is as follows: Much like 18, each processors p_i of the n processors looks at its corresponding integer x_i and confirms that x_i is in the input by writing a 1 to the (initially 0) memory location $IsInInput[i]$ i.e. when $IsInInput[i] = 1$, the integer i is in the input and when $IsInInput[i] = 0$, the integer i is not in the input. As in 18, this step takes constant time because each processor works on one integer and performs one read step and one write step.

Next, each processor p_i reads $IsInInput[i]$ and, if p_i finds a 1, it writes a 1 to the array $RootChunks[\lceil \frac{i}{\sqrt{n}} \rceil - 1]$, which is an array of size $\sqrt{n}$ initialized to all 0s. If $RootChunks[k] = 1$, then there is a 1 within the kth $\sqrt{n}$-sized chunk of $IsInInput$ i.e. one of the integers from $k * \sqrt{n}$ to $(k+1) * \sqrt{n}$ is present in the input. If $RootChunks[k] = 0$, then there is not a 1 within the kth $\sqrt{n}$-sized chunk of $IsInInput$ i.e. none of the integers from $k * \sqrt{n}$ to $(k+1) * \sqrt{n}$ are present in the input. This step takes constant time because each processor works on one integer and performs one read step and one write step to one of four locations. Each processor writes a 1, if anything, to one of four locations, so it does not violate the CRCW Common restriction.

Next, the n processors find the max index of $RootChunks$ that contains a 1. That is, we use n processors to find the

max of the $\leq \sqrt{n}$ indices that contain a 1. This subroutine is referred to in the algorithm as $IndicesContainingOne$. The result of this step, m, corresponds to the mth $\sqrt{n}$-size chunk of $IsInInput$ - this chunk contains the largest value of the input array.

Finally, the n processors then find the max index of the mth $\sqrt{n}$-size chunk of $IsInInput$ that contains a 1. The resulting index is the max of the input array $x_1, ..., x_n$. Both this step and the prior step take constant time, as for both we are using $n = j^2$ processors to compute the max of $\sqrt{n} = j$ integers, which takes $O(1)$ time.

We have four steps that each take constant time, so this algorithm is $O(1)$ on a CRCW Common PRAM.

Algorithm 2 CRCW Common $O(1)$ algorithm for maximum with n processors.

Require: Input $x_1...x_n$, an n-sized memory location $IsInInput$, a $\sqrt{n}$-sized memory location $RootChunks$, a memory location $MaxIndex$ and a memory location $Maximum$.

$IsInInput[i] \leftarrow 0$ ▷ First, zero out the $IsInInput$ array.

$IsInInput[x_i] \leftarrow 1$ ▷ This is a CRCW Common step, since all processors write the same number: 1.

if $IsInInput[i] == 1$ **then**

$RootChunks[\lceil \frac{i}{\sqrt{n}} \rceil - 1] \leftarrow 1$ ▷ This is a CRCW Common step.

end if

$MaxIndex \leftarrow Max(IndicesContainingOne(RootChunks))$ ▷ The Max algorithm from problem 17

$Maximum \leftarrow Max(IndicesContainingOne(IsInInput[MaxIndex * \sqrt{n}]$ to $IsInInput[MaxIndex * \sqrt{n}]))$

20

Its clear that this algorithm would be incredibly trivial if the machine were not limited to exclusive write. Each processor would simply read its corresponding B value, and write that index in A to its corresponding index in C. However, due to the data bottleneck that occurs with A, we must make copies.

The algorithm begins with each processor making $log^2(n)$ copies of its corresponding index in A. These copies are put into a $log^2(n) \times n$ array.