Two Interesting Problems

Question 1: We define the following two operators on an array $[a_0, a_1, \ldots, a_{n-1}]$ of integers**:**

- PRESCAN(A) returns $[0, a_0, a_0+a_1, a_o+a_1+a_2, \ldots, a_0+a_1+\ldots+a_{n-2}]$,
- SCAN(A) returns $[a_0, a_0+a_1, a_0+a_1+a_2, \ldots, a_0+a_1+\ldots+a_{n-1}]$,

We have already seen how to implement these two operators in time O(log n) on an **EREW PRAM**. Consider the SPLIT function below:

```
1  SPLIT(A, Flags)
2      Idown ← PRESCAN(not(Flags))
3      Iup ← n - REVERSE(SCAN(REVERSE(Flags)))
4      forall i ∈ {1, ..., n} in parallel do
5          if Flags(i) then Index[i] ← Iup[i]
6          else Index[i] ← Idown[i]
7      Result ← PERMUTE(A, Index)
8      return Result
```

This function uses two functions: REVERSE(A) and PERMUTE$(A, Index)$. The former reverses array A, and the latter reorders array A according to a permutation specified as an array of indices, $Index$. The slightly cumbersome REVERSE(SCAN(REVERSE($Flags$))) simply scans from the end of the array $Flags$, considering its elements as integers.

1. Given an array *Flags* of booleans, what does the SPLIT function returns? What is its execution time?

Using $O(n)$ PUs SCAN and PRESCAN can be done in time $O(\log n)$. As other operations only require constant time, SPLIT runs in time $O(\log n)$.

Answer:

▷ **Question 1.** Let us look at an example:

$$\begin{array}{rl} A & = [\ 5\ \ 7\ \ 3\ \ 1\ \ 4\ \ 2\ \ 7\ \ 2\ ] \\ Flags & = [\ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ ] \\ Idown & = [\ 0\ \ 0\ \ 0\ \ 0\ \ \boxed{0}\ \ \boxed{1}\ \ 2\ \ \boxed{2}\ ] \\ Iup & = [\ \boxed{3}\ \ \boxed{4}\ \ \boxed{5}\ \ \boxed{6}\ \ 7\ \ 7\ \ \boxed{7}\ \ 8\ ] \\ Index & = [\ 3\ \ 4\ \ 5\ \ 6\ \ 0\ \ 1\ \ 7\ \ 2\ ] \\ Result & = [\ 4\ \ 2\ \ 2\ \ 5\ \ 7\ \ 3\ \ 1\ \ 7\ ] \end{array}.$$

With this example it is easy to see what function SPLIT does: elements of A whose corresponding elements in *Flags* are equal to 0 are moved to the beginning of *Result*, remaining in the same order. Likewise, elements of A whose corresponding elements in FLAGS are equal to 1 are grouped at the end of *Result*, remaining in the same order.

Using $O(n)$ PUs SCAN and PRESCAN can be done in time $O(\log n)$. As other operations only require constant time, SPLIT runs in time $O(\log n)$.

Question 2:

2. Consider the following MYSTERY function:

```
1  MYSTERY(A, Number_Of_Bits)
2      for i = 0 to Number_Of_bits − 1 do
3          bit(i) ←array indicating whether the i^th bit of
           elements of A is equal to 1 or not
4          A ←SPLIT(A, bit(i))
```

(a) What is the result of the MYSTERY function when applied to $A = [5, 7, 3, 1, 4, 2, 7, 2]$ and $Number_Of_Bits = 3$?

(b) What does the MYSTERY function compute?

(c) Assuming the size of integers is $O(\log n)$ bits, what is the execution time of MYSTERY with n PUs? What if only p PUs are used? What are the values of p that lead to an optimal value of the algorithm's work?

▷ **Question 2.**

(a)

A	$=$	[5	7	3	1	4	2	7	2]
$bit(0)$	$=$	[1	1	1	1	0	0	1	0]
$A \leftarrow$ Split$(A, bit(0))$	$=$	[4	2	2	5	7	3	1	7]
$bit(1)$	$=$	[0	1	1	0	1	1	0	1]
$A \leftarrow$ Split$(A, bit(1))$	$=$	[4	5	1	2	2	7	3	7]
$bit(2)$	$=$	[1	1	0	0	0	1	0	1]
$A \leftarrow$ Split$(A, bit(2))$	$=$	[1	2	2	3	4	5	7	7]

(b) Based on the example, it looks like Mystery sorts its input array. In fact, it is a parallel implementation of the well-known *radix-sort* algorithm: starting with the least-significant bit, the Split function splits the array in two parts depending on the value of this bit. Each call to Split sorts elements according to the current bit value while maintaining the order obtained with previous bits. This is why the algorithm goes from the least-significant bit to the most-significant bit.

(c) There are $O(\log n)$ iterations of the main loop. The execution time of the Mystery function is thus $O(\log^2 n)$ with $O(n)$ PUs. When using only p PUs, the execution time of Split becomes $O(\frac{n}{p} + \log p)$ and the execution time of the parallel radix-sort becomes $O((\frac{n}{p} + \log p) \log n) = O(\frac{n}{p} \log n + \log n \log p)$. The work is optimal (i.e., equal to $O(n \log n)$) for p such that $p \log p \leqslant n$, e.g., for $p = n^q$ with $0 < q < 1$.