Exercice 1

Do the following former exam. See last pages

Exam Parallelism Fabrice

For i=1 to N do if T[i]=1 return i

// there is a ~~0~~1 in the array !

T = O(N)

Firstone Array B.

prelim = For i=1 to N in parallel do each

B[i] = A[i]

modif . For all i, j such that i<j in //

if B[i]=1 & B[j]=1

then B[j] = 0

Q) if B[i] and B[j] are =1 tog this means that B[j] /(A[j]) is not the first one occurence

So at the end of the ~~for~~ all loop,

if A[k] = B[k] = 1 , there is not any l < k such that A[l]=1, so A[k] is the cell having the first 1

And, in all other B[x] having A[x]=1, there is now a 0

Write in Pram.

for each i, j, i<j do in parallel

// time = O(1) if (B[i] = 1 & B[j] = 1)

then B[j] = 0

while a proc reads a cell, eg B[l]

another may read also that same cell B[k]

where the k may be = to the l.

So CR needed , O(n²) proc indeed

only $\frac{(N)(N+1)}{2}$

1 2 3 4 .. N

2 3 4 N

3 4 .. N

(N-1)(N)

CW arbitrary

writing 1

if a write

op needed

$N + (N-1) + (N-2) ... + 1 = \sum_{i=1}^{N} i = \frac{N \times (N+1)}{2}$

A =

0 0 0 1 0 1 0 1

1 2 3 4 5 6 7 8

Ex i, j

(4, 5)

(4, 8)

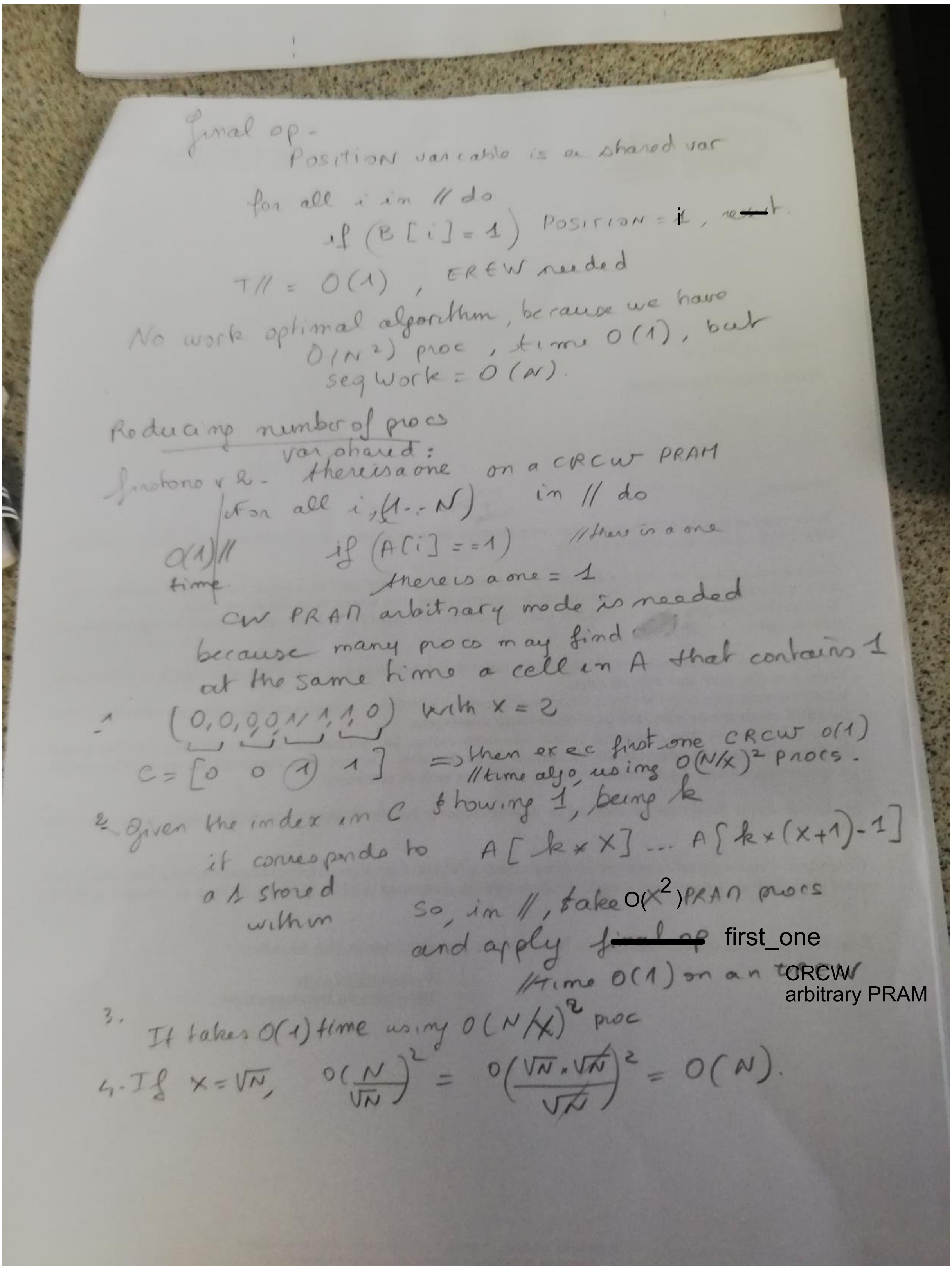
ou

4, 8 j = 8.

6, 8

B[8] = 0

write en conc.



final op -

Position variable is a shared var

for all i in // do

if $(B[i] = 1)$ POSITION = i , [redacted]

$T// = O(1)$, EREW needed

No work optimal algorithm, because we have $O(N^2)$ proc , time $O(1)$, but seq Work = $O(N)$.

Reducing number of procs

firstone v 2 - thereisaone on a CRCW PRAM

var shared :

for all i, (1 .. N) in // do

$O(1)$ // time

if $(A[i] == 1)$ //there is a one

thereis a one = 1

CW PRAM arbitrary mode is needed because many procs may find at the same time a cell in A that contains 1

1\. (0,0,0,0,1,1,1,0) with x = 2

C = [0 0 1 1] => then exec first_one CRCW $O(1)$ //time also, using $O(N/x)^2$ procs.

2\. Given the index in C showing 1, being k it corresponds to $A[k \times X] \dots A[k \times (X+1) - 1]$ a 1 stored within

So, in //, take $O(X^2)$ PRAM procs and apply [redacted] first_one //time $O(1)$ on an CRCW arbitrary PRAM

3\. It takes $O(1)$ time using $O(N/X)^2$ proc

4\. If $x = \sqrt{N}$, $O\left(\frac{N}{\sqrt{N}}\right)^2 = O\left(\frac{\sqrt{N} \cdot \sqrt{N}}{\sqrt{N}}\right)^2 = O(N)$.

Exercice 2

Goal is to find what is the algorithm used below to sort in an increasing way using a PRAM. The expected behaviour is summarized by the figure below.

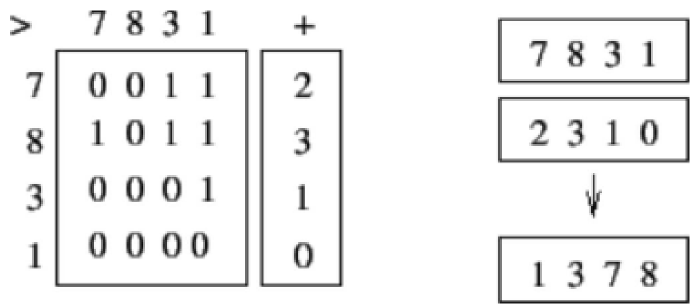


Figure: Exemple de tri sur la séquence de valeurs 7;8;3;1

Sketch the algorithm, with the aim it runs as quick as possible, without constraint on the number of required processors of the PRAM. Give precisely which PRAM variant is needed, again, do not constraint yourself. Under the necessary PRAM variant to use, give precisely what is the parallel time, and the work. To which sort of sequential sorting method does that algorithm belong to ?

We proceed like this; the list (vector) of values to sort, name it L, is put as a line, and that same data is transposed and put as a column, let it name C. Then, in parallel, each pair of values formed by one value from the line, one value from the column is considered. If the value in the line is less than the value in the column 1 is written, otherwise 0 is written in the matrix entry corresponding to the pair. Eg, when we compare 3 (stored in L[2]) with 8 (stored in C[1]), we will write 1 in M[2,1] (ie C[1]=8 has one more element standing before it) .
After that, all values from each a single line of the matrix are summed and this creates into a new vector V. These sums give respective position of values in the initial vector (C) in the resulting sorted vector.

So we need O(N^2) procs of the PRAM for filling the NxN matrix in parallel (one processor per pair, except the diagonal, so precisely, we need N^2 – N procs. If we assume that the list to sort, has been copied to stand as the column of the matrix, the read operations still need a CR PRAM, because, as for max-v1 -see TD1-, a proc in charge of the pair (i,j) needs to read L[x] with x=i, while another proc in charge of the pair (i,k)   may read L[x] with x=i too.
Once the matrix has been filled with 0 and 1, we need to get the sum for each line, to be stored in the corresponding vector V entry. This can run using a EREW PRAM sum-reduce , with O(N) processors working in parallel *for each* matrix line, and sum these values of a line in parallel time O(log N). As we have N lines, we need to enroll NxN processors. The last stage only requires to enroll N processors, each being in charge of reading V[i], in order to decide where to move L[i] in the final ordered list say L', more precisely, L'[V[i]]=L[i].
EREW is needed here.

To sum up, this sorting algorithm runs on a CREW PRAM, in parallel time O(log N), using O(N^2) processors. The work is high! It is clearly not a work optimal solution.
It seems to me that it is a method like insertion sort, which is based on comparisons between pairs of elements. https://en.wikipedia.org/wiki/Sorting_algorithm. A more precise reading tells us that it is a counting sorting algorithm.
Notice that this parallel algorithm does *not* belong to the family of "comparator exchange elements"-based sorting networks.