PRAM computation model

PRAM computation model

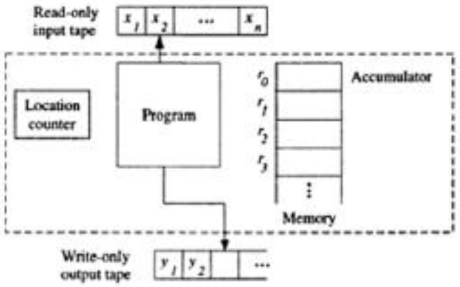
To abstract away parallel machines, and being able to compare parallel algorithms

RAM: A MODEL OF SERIAL COMPUTATION

The Random Access Machine (RAM) is a model of a one-address computer.

- Consists of a memory
- A read-only input tape
- A write-only output tape
- A program

Input tape consists of a sequence of integers.
Every time an input value is read, the input head advances one square.
Likewise, the output head advances after every write.



Aho, HopCroft, and Ulman, 1974

Memory consists of unbounded set of registers, r_0, r_1, …

Each register holds a single integer.

Register r_0 is the accumulator, where computations are performed.

COST MODELS

Uniform Cost Criterion: each RAM instruction requires one unit of time to execute. Every register requires one unit of space.

Logarithmic Cost Criterion: Assumes that every instruction takes a logarithmic number of time units (wrt. the length of the operands), and that every register requires a logarithmic number of units of space.

Thus, uniform cost criteria count the number of operations and logarithmic cost criteria count the number of bit operations.

The uniform cost criterion is applicable if the values manipulated by a program always fit into one computer word.

Consider an 8 bit adder.
In the uniform cost criteria to analyze the run time of the adder, we would say that the adder takes 1 unit of time, ie. $T(N)=1$.
However, in the logarithmic model you would consider that the 1's position bits are added, followed by the 2's position bits, and so on. In this model, thus there are 8 smaller additions (for every bit positions) and each requires a unit of time. Thus, $T(N)=8$. Generalizing, $T(N)=\log(N)$.

TIME COMPLEXITIES IN THE RAM MODEL

Worst case time complexity: The function f(n), the maximum time taken by the program to execute over all inputs of size n.

Expected time complexity: It is the average time over the execution times for all inputs of size n.

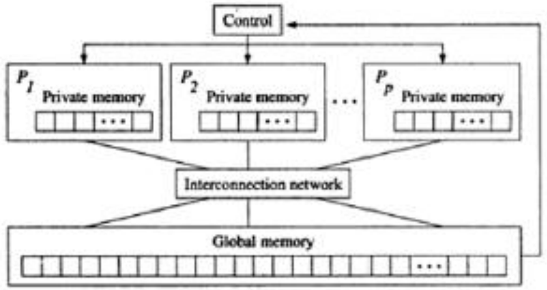
Analogous definitions hold for the space complexities (just replace the time word by space).

THE PRAM MODEL

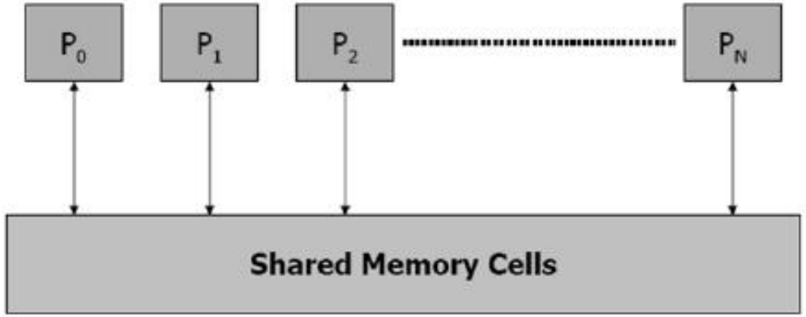
A PRAM consists of a control unit, global memory, an unbounded set of processors, each with its own private memory.

Active processors execute identical instructions.

Every processor has a unique index, and the value can be used to enable or disable the processor, or influence which memory locations it accesses.



A SIMPLISTIC PICTURE



Cost of a PRAM computation is the product of the parallel time complexity and the number of processors used. For example, a PRAM algorithm that has time complexity $\Theta(\log p)$ using p processors has cost $\Theta(p \log p)$.

- All processing elements (PE) execute synchronously the same algorithm and work on distinct memory areas.
- Neither the number of PEs nor the size of memory is bounded.
- Any PE can access any memory location in one unit of time.
    - The last two assumptions are unrealistic!

THE PRAM COMPUTATION STEPS

A PRAM computation starts with the input stored in global memory and a single active processing element.

During each step of the computation an active, enabled processor may read a value from a single private or global memory location, perform a single RAM operation, and write into one local or global memory location.

Alternatively, during a computation step a processor may activate another processor.

All active, enabled processors must execute the same instruction, albeit on different memory locations.

- This condition can be relaxed. However we will stick to it.

The computation terminates when the last processor halts.

PRAM MODELS

The models differ in how they handle read or write conflicts, ie. when two or more processors attempt to read from or write to the same global memory location.

1. EREW (Exclusive Read Exclusive Write) Read or write conflicts are not allowed.
2. CREW (Concurrent Read Exclusive Write) Concurrent reading allowed, ie. Multiple processors may read from the same global memory location during the same instruction step. Write conflicts are not allowed.
   1. During a given time, ie. During a given step of an algorithm, arbitrarily many PEs can read the value of a cell simultaneously while at most one PE can write a value into a cell.
3. CRCW (Concurrent Read Concurrent Write): Concurrent reading and writing are allowed. A variety of CRCW models exist with different policies for handling concurrent writes to the same global address:
   1. Common: All processors concurrently writing into the same global address must be writing the same value.
   2. Arbitrary: If multiple processors concurrently write to the same global address, one of the competing processors is arbitrarily choses as the winner, and its value is written.
   3. Priority: The processor with the lowest index succeeds in writing its value.

RELATIVE STRENGTHS

The EREW model is the weakest.

- A CREW PRAM can execute any EREW PRAM algorithm in the same time. This is obvious, as the concurrent read facility is not used.
- Similarly, a CRCW PRAM can execute any EREW PRAM algorithm in the same amount of time.

The PRIORITY PRAM model is the strongest.

- Any algorithm designed for the COMMON PRAM model will execute in the same time complexity in the ARBITRARY or PRIORITY PRAM models.
  - If the processors writing to the same location write the same value choosing an arbitrary processor would cause the same result.
  - Likewise, it also produces the same result when the processor with the lowest index is chosen the winner.

Because the PRIORITY PRAM model is stronger than the EREW PRAM model, an algorithm to solve a problem on the EREW PRAM can have higher time complexity than an algorithm solving the same problem on the PRIORITY PRAM model.

COLE'S RESULT ON SORTING ON EREW PRAM

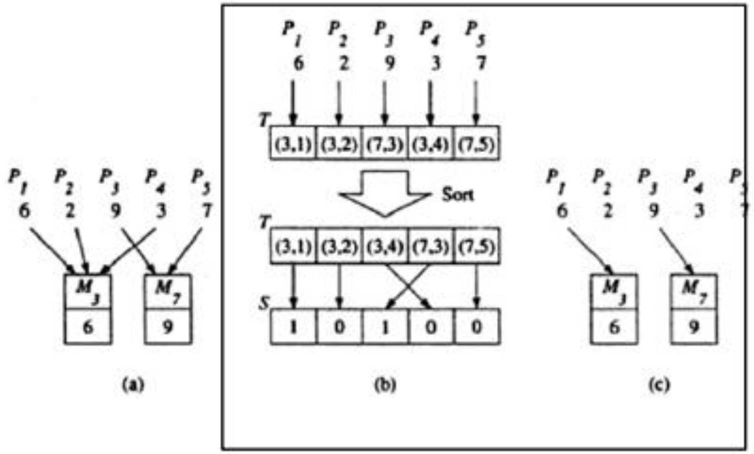
Cole [1988] A p-processor EREW PRAM can sort a p-element array stored in global memory in $\Theta(\log p)$ time.

How can we use this to simulate a PRIORITY CRCW PRAM on an EREW PRAM model?

SIMULATING PRIORITY-CRCW ON EREW

Concurrent write operations take constant time on a p-processor PRIORITY PRAM.

a) Processors P1, P2, P4 attempt to write values to memory locations M3. P1 wins, as it has least index. P3 and P5 attempts to write at M7. P3 wins.



Processor P1 reads memory location T1, retrieves (3,1) and writes 1 to S1. P2 reads T2, ie. (3,2), and then reads T1 ie. (3,1). Since the first arguments match, it flags S2=0. Likewise for the rest. **Thus the highest priority processor accessing any particular location can be found in constant time.**
Finally, the winning processors write their values.

b) Simulating Concurrent write on the EREW PRAM model.
Each processor writes (address,processor number) to a global array T.
The processors sort T in $\Theta(\log p)$.
In constant time, the processors can set 1 in those indices in S which corresponds to winning processors.

IMPLICATION

A p-processor PRIORITY PRAM can be simulated by a p-processor EREW PRAM with time complexity increased by a factor of $\Theta(\log p)$.

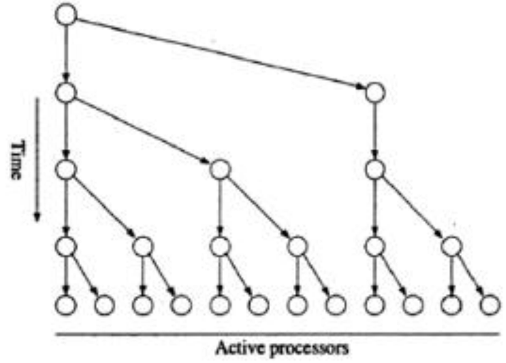
PRAM ALGORITHMS

PRAM algorithms work in two phases:

First phase: a sufficient number of processors are activated.

Second phase: These activated processors perform the computation in parallel.

Given a single active processor to begin with it is easy to see that $\lceil \log p \rceil$activation steps are needed to activate p processors.



Meta-Instruction in the PRAM algorithms:
spawn (<processor names>)
To denote the logarithmic time activation of processors from a single active processor.

SECOND PHASE OF PRAM ALGORITHMS

To make the programs of the second phase of the PRAM algorithms easier to read, we allow references to global registers to be array references.

We assume there is a mapping from these array references to appropriate global registers.

The construct

for all <processor list> do <statement list> endfor

denotes a code segment to be executed **in parallel** by all the specified processors.

Besides the special constructs already described, we express PRAM algorithms using familiar control constructs: if…then….else…endif, for…endfor, while…endwhile, and repeat…until. The symbol $\leftarrow$ denotes assignment.

PARALLEL REDUCTION

The binary tree is one of the most important paradigms of parallel computing.

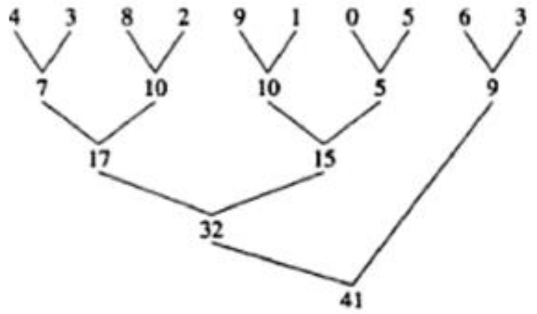
In the algorithms that we refer here, we consider an inverted binary tree.

- Data flows from the leaves to the root. These are called fan-in or reduction operations.

More formally, given a set of n values a1, a2, …, an and an associative binary operator $\oplus$, reduction is the process of computing $a1 \oplus a2 \oplus \cdots \oplus an$.

- Parallel Sum is an example of a reduction operation.

PARALLEL SUMMATION IS AN EXAMPLE OF REDUCTION



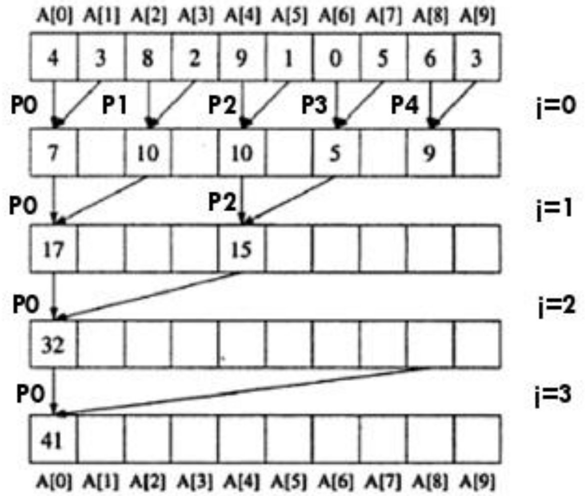
How do we write the PRAM algorithm for doing this summation?

GLOBAL ARRAY BASED EXECUTION

The processors in a PRAM algorithm manipulate data stored in global registers.

For adding n numbers we spawn $\lfloor\left(\frac{n}{2}\right)\rfloor$ processors.

Consider the example to generalize the algorithm.


A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9]
4 3 8 2 9 1 0 5 6 3
P0 P1 P2 P3 P4
j=0
7 10 10 5 9
P0 P2
j=1
17 15
P0
j=2
32
P0
j=3
41
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9]

GLOBAL ARRAY BASED EXECUTION

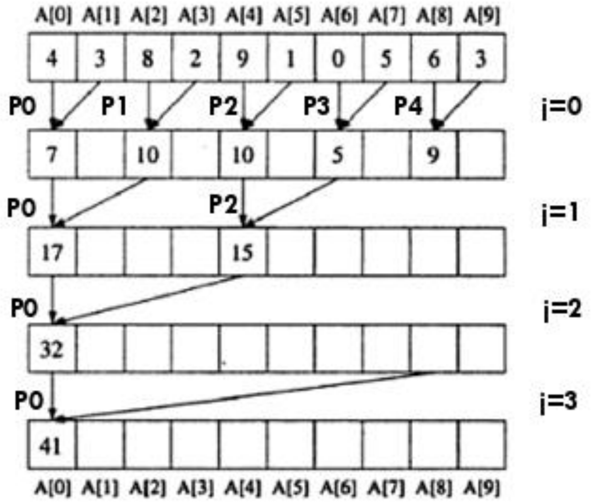
Each addition corresponds to:

$A[2i]+A[2i+2^i]$.

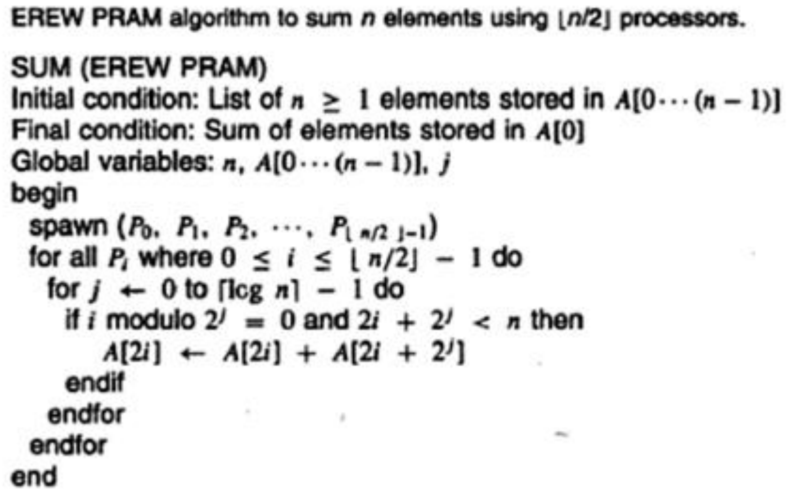
Note, the processor which is active has an i such that:
$i \bmod 2^i=0$ (ie. keep only those processors active).

Also check that the array does not go out of bound.

- ie, $2i+2^i<n$


A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9]
4 3 8 2 9 1 0 5 6 3
P0 P1 P2 P3 P4
i=0
7 10 10 5 9
P0 P2
i=1
17 15
P0
i=2
32
P0
i=3
41
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9]

EREW PRAM PROGRAM



EREW PRAM algorithm to sum n elements using $\lfloor n/2 \rfloor$ processors.

```
SUM (EREW PRAM)
Initial condition: List of $n \geq 1$ elements stored in $A[0 \cdots (n-1)]$
Final condition: Sum of elements stored in $A[0]$
Global variables: $n$, $A[0 \cdots (n-1)]$, $j$
begin
  spawn ($P_0, P_1, P_2, \cdots, P_{\lfloor n/2 \rfloor - 1}$)
  for all $P_i$ where $0 \leq i \leq \lfloor n/2 \rfloor - 1$ do
    for $j \leftarrow 0$ to $\lceil \lg n \rceil - 1$ do
      if $i$ modulo $2^j = 0$ and $2i + 2^j < n$ then
          $A[2i] \leftarrow A[2i] + A[2i + 2^j]$
      endif
    endfor
  endfor
end
```

COMPLEXITY

The SPAWN routine requires $\lceil log \left\lfloor \frac{n}{2} \right\rfloor \rceil$ doubling steps.

The sequential for loop executes $\lceil \log n \rceil$ times.

- Each iteration takes constant time.

Hence overall time complexity is $\Theta(\log n)$ given n/2 processors.

PREFIX SUM

Given a set of n values a1, a2, …, an, and an associative operation $\oplus$, the prefix sum problem is to calculate the n quantities:

a1,

a1 $\oplus$ a2,

…

a1 $\oplus$ a2 $\oplus$ … $\oplus$ an

AN APPLICATION OF PREFIX SUM

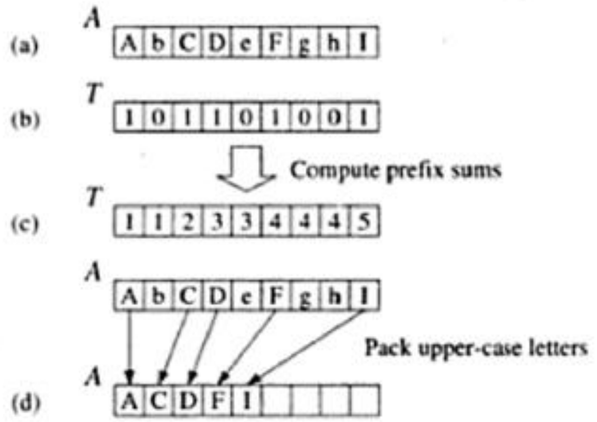
We are given an array A of n letters. We want to pack the uppercase letters in the initial portion of A while maintaining their order. The lower case letters are deleted.

a) Array A contains both uppercase and lowercase letters. We want to pack uppercase letters into beginning of A.

b) Array T contains a 1 for every uppercase letter, and 0 for lowercase.

c) Array T after prefix sum. For every element of A containing an uppercase letter, the corresponding element of T is the element's index in the packed array.

d) Array A after packing.

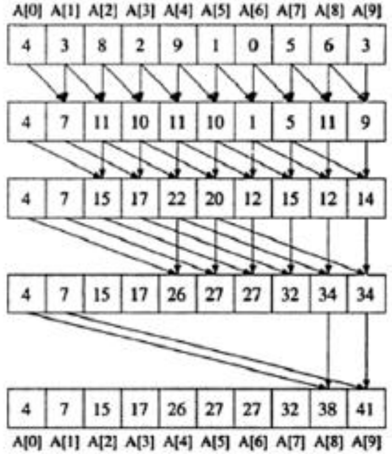


GLOBAL ARRAY BASED EXECUTION IN EREW

There are n-1 processors activated.

Each one accesses A[i], then accesses A[i-2^j], where j is the depth (j varies from 0 to $\lceil \log n \rceil - 1$.

Of course, the bounds need to be checked.



THE PRAM PSEUDOCODE

```
PREFIX.SUMS (CREW PRAM):
Initial condition: List of $n \geq 1$ elements stored in $A[0 \cdots (n-1)]$
Final condition: Each element $A[i]$ contains $A[0] \oplus A[1] \oplus \cdots \oplus A[i]$
Global variables: $n$, $A[0 \ldots (n-1)]$, $j$
begin
 spawn ($P_1$, $P_2$, ..., $P_{n-1}$)
 for all $P_i$ where $1 \leq i \leq n-1$ do
  for $j \leftarrow 0$ to $\lceil \log n \rceil - 1$ do
   if $i - 2^j \geq 0$ then
     $A[i] \leftarrow A[i] + A[i - 2^j]$
   endif
  endfor
 endfor
end
```

COMPLEXITY

Running time is $t(n) = O(\lg n)$

Cost is $c(n) = p(n) \times t(n) = O(n \lg n)$

Note not cost optimal, as RAM takes $O(n)$

MAKING THE ALGORITHM COST OPTIMAL

Example Sequence – 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Use $n / \lceil \lg n \rceil$ PEs with lg(n) items each

0,1,2,3   4,5,6,7   8,9,10,11   12,13,14,15

STEP 1: Each PE performs sequential prefix sum

0,1,3,6   4,9,15,22   8,17,27,38   12,25,39,54

STEP 2: Perform parallel prefix sum on last nr. in PEs

0,1,3,6   4,9,15,28   8,17,27,66   12,25,39,120

Now prefix value is correct for last number in each PE

STEP 3: Add last number of each sequence to incorrect sums in next sequence (in parallel)

0,1,3,6   10,15,21,28   36,45,55,66   78,91,105,120

A COST-OPTIMAL EREW ALGORITHM

In order to make the prefix algorithm optimal, we must reduce the cost by a factor of lg n.

We reduce the nr of processors by a factor of lg n (and check later to confirm the running time doesn't change).

Let $k = \lceil \lg n \rceil$ and $m = \lceil n/k \rceil$

The input sequence $X = (x_0, x_1, \ldots, x_{n-1})$ is partitioned into m subsequences $Y_0, Y_1, \ldots, Y_{m-1}$ with k items in each subsequence.

- While Y_{m-1} may have fewer than k items, without loss of generality (WLOG) we may assume that it has k items here.

Then all sequences have the form,

$$Y_i = (x_{i*k}, x_{i*k+1}, \ldots, x_{i*k+k-1})$$

PRAM ALGORITHM OUTLINE

Step 1: For $0 \leq i < m$, each processor P_i computes the prefix computation of the sequence $Y_i = (x_{i*k}, x_{i*k+1}, \ldots, x_{i*k+k-1})$ using the RAM prefix algorithm (using $\oplus$) and stores prefix results as sequence $s_{i*k}, s_{i*k+1}, \ldots, s_{i*k+k-1}$.

Step 2: All m PEs execute the preceding PRAM prefix algorithm on the sequence $(s_{k-1}, s_{2k-1}, \ldots, s_{n-1})$

- Initially P_i holds s_{i*k-1}
- Afterwards P_i places the prefix sum $s_{k-1} \oplus \ldots \oplus s_{ik-1}$ in s_{ik-1}

Step 3: Finally, all P_i for $1 \leq i \leq m-1$ adjust their partial value sums for all but the final term in their partial sum subsequence by performing the computation

$$s_{ik+j} \leftarrow s_{ik+j} \oplus s_{ik-1}$$

for $0 \leq j \leq k-2$.

COMPLEXITY ANALYSIS

Analysis:

- Step 1 takes O(k) = O(lg n) time.
- Step 2 takes O(lg m) = O(lg n/k)

  = O(lg n- lg k) = O(lg n - lg lg n)

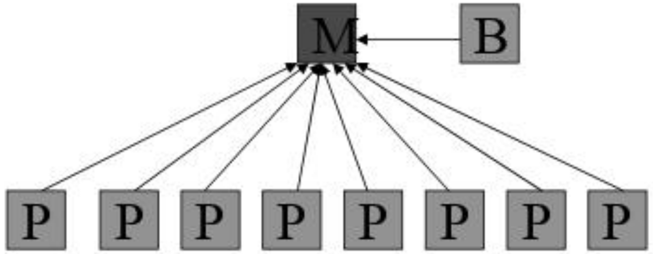
  = O(lg n)
- Step 3 takes O(k) = O(lg n) time
- The running time for this algorithm is O(lg n).
- The cost is O((lg n) × n/(lg n)) = O(n)
- Cost optimal, as the sequential time is O(n)

Can you write the complete pseudocode in the PRAM model?

BROADCASTING ON A PRAM

"Broadcast" can be done on CREW PRAM in O(1) steps:

- Broadcaster sends value to shared memory
- Processors read from shared memory



Requires lg(P) steps on EREW PRAM.

CONCURRENT WRITE – FINDING MAX

Finding max problem

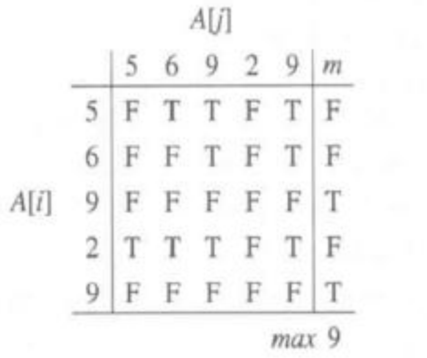
- Given an array of n elements, find the maximum(s)
- sequential algorithm is O(n)

Data structure for parallel algorithm

- Array A[1..n]
- Array m[1..n]. m[i] is true if A[i] is the maximum
- Use n^2 processors

```
Fast_max(A, n)
1.for i = 1 to n do, in parallel
2.    m[i] = true        // A[i] is potentially maximum
3.for i = 1 to n, j = 1 to n do, in parallel
4.    if A[i] < A[j] then
5.        m[i] = false
6.for i = 1 to n do, in parallel
7.    if m[i] = true then max = A[i]
8.return max
```

Time complexity: O(1)



		A[j]					
		5	6	9	2	9	m
	5	F	T	T	F	T	F
	6	F	F	T	F	T	F
A[i]	9	F	F	F	F	F	T
	2	T	T	T	F	T	F
	9	F	F	F	F	F	T
						max	9

CONCURRENT WRITE – FINDING MAX

Concurrent-write

- In step 4 and 5, processors with A[i] < A[j] write the same value 'false' into the same location m[i]
- This actually implements $m[i] = (A[i] \geq A[1]) \wedge \ldots \wedge (A[i] \geq A[n])$

Is this work efficient?

- No, n^2 processors in O(1)
- $O(n^2)$ work vs. sequential algorithm is O(n)

What is the time complexity for the Exclusive-write?

- Initially elements "think" that they might be the maximum
- First iteration: For n/2 pairs, compare.
  - n/2 elements might be the maximum.
- Second iteration: n/4 elements might be the maximum.
  - log n th iteration: one element is the maximum.
- So Fast_max with Exclusive-write takes O(log n).

O(1) (CRCW) vs. O(log n) (EREW)

CRCW VERSUS EREW - DISCUSSION

CRCW

- Hardware implementations are expensive
- Used infrequently
- Easier to program, runs faster, more powerful.
- Implemented hardware is slower than that of EREW
  - In reality one cannot find maximum in O(1) time

EREW

- Programming model is too restrictive
  - Cannot implement powerful algorithms

So, CREW is the most popular parallel model.