PRAM Algorithms

Prepared by:
Dr. Rashmi S

PRAM Algorithms

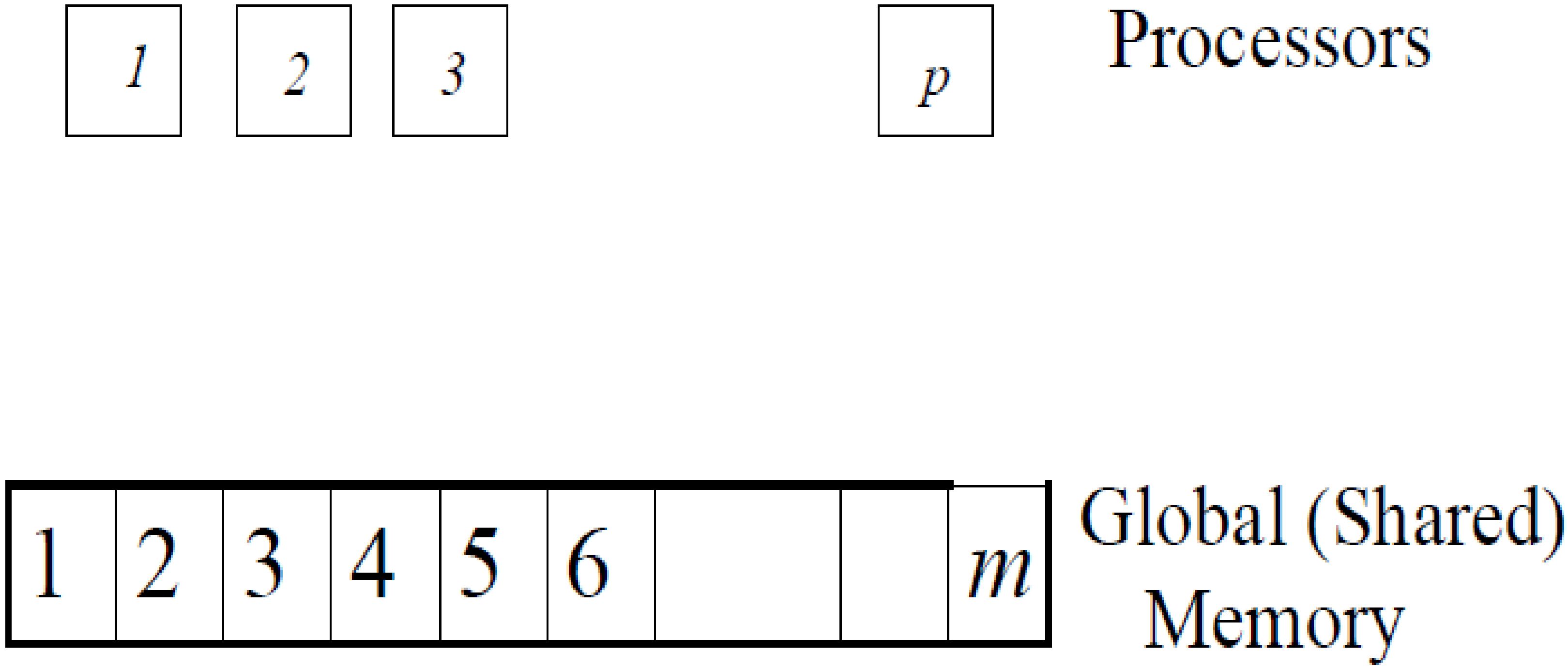
- Parallel machines – computers with more than one processor
- Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena
- Ex- weather forecasting , Medical imaging and diagnosis

1 Computational Model

- Most popular theoretical Model : PRAM (parallel random-access machine)
  - X processors, P_0, P_1, …, P_{x-1} share global memory (or shared memory) and a common clock
  - May execute different instructions at the same time, e.g. read/write at the same time
  - Key assumption : running time can be measured as the number of parallel memory accesses

  i.e. In *1*-cpu machine, to do 1 statement, say, 1 unit of time.
  In *p*-cpu machines, to do 1 statement (each cpu in parallel), also, 1 unit of time

A Parallel Random Access Machine (PRAM)



Possible Read and write Conflict !

- **Variants of PRAM algorithms**
  - Concurrent Read (CR) : may read same location at the same time
  - Exclusive Read (ER) - no two processors can read same location at the same time
  - Also, Concurrent Write (CW) and Exclusive Write (EW)
  - Read from and write to same location at the same time is not allowed
  - Types of algorithms : EREW, CREW, ERCW, CRCW

    Not many algorithms on ERCW

    A PRAM that supports EREW is called EREW-PRAM

    (similarly, we can define CRCW-PRAM etc)

- Most algorithms assume n, log n, or n/lg n number of processors. In practice, this is not a realistic assumption.
- For CW, assume all CPU write the same value.

  - common CRCW PRAM – permitted only if all have the same message to write

  - arbitrary CRCW PRAM – one of them will be successful (write)

  - priority CRCW PRAM – the one with the highest priority

Simple CRCW algorithm :

To compute a[0] = a[1] || a[2] || … || a[n]

Boolean (or logical) OR of the n bits

a[1:n] -- a[1], a[2], … , a[n]

```
for each processor i ( 1 <= i <= n) in parallel
    if (A[i] = 1) then A[0] := A[i];
```

The boolean OR of n bits can be computed in O(1) time on an n-processor common CRCW PRAM.

- Running time analysis : For a given problem X with input size n,

  Let the run time of a parallel algorithm using p processors be $T(n,p)$

  Let the run time of a best known sequential algorithm be $S(n)$

  - The total work of a parallel algorithm is :

    $$p * T(n,p)$$

  - The speedup of a parallel algorithm is

    $$S(n)/T(n,p)$$

- A parallel algorithm is said to be work-optimal if $p * T(n,p) = O(S(n))$

- A parallel algorithm is work-optimal if and only if it has linear speedup

- For above example(Logical OR):
  - $S(n) = O(n)$, $n*T(n,n) = n*O(1)$ → work optimal.

- If speedup is $O(p)$ then the algorithm is said to have a linear speedup

2 Prefix computation

- Let $\oplus$ = binary, associative operator, i.e. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- Example : +, -, *, AND, OR etc
- Problem : Given n elements, $x_1, x_2, \ldots, x_n$. Compute n elements x_1, $x_1 \oplus x_2, \ldots,$ $x_1 \oplus x_2 \oplus x_3 \ldots x_{n-1} \oplus x_n$

- Algorithm : using n CPUs; assume n is 2^k

for each CPU_i in parallel /* initialize */

$y[i] = x_i$

for i = 0 to k-1 do

for each CPU_j where $j > 2^i$ in parallel

$y[j]=y[j] \oplus y[j-2^i]$

- Example : input <3 1 4 5 2 3 6 7>

	index	1	2	3	4	5	6	7	8
initial	y[]	***3***	1	4	5	2	3	6	7
i = 0	y[]		***4***	5	9	7	5	9	13
i = 1	y[]			***8***	***13***	12	14	16	18
i = 2	y[]					***15***	***18***	***24***	***31***

- Analysis
  - Above algorithm is EREW (or CREW) algorithm
  - The run time of best sequential algorithm is $O(n)$

    The run time of above algorithm is, $T(n,n) = O(\lg n)$

    The total work is $O(n \lg n)$

    It is not work optimal!

- To get work optimal algorithm, we need to use only $(n/\log n)$ CPUs

- Work optimal algorithm : using (n/log n) CPUs

1. Assign log n elements to each CPU
2. Each CPU computes the prefixes of its assigned log n elements using a simple sequential algorithm
3. From Step 2, use last element from each CPU, i.e. total (n/log n) elements.

   Use (n/log n) CPUs to compute prefixes of theses n/log n elements using previous parallel algorithm (see example, record results in a new array)
4. Each CPU updates its log n elements using results from Step 3

- Example : input <3 1 4 5 2 3 6 7>

	CPU 1			CPU 2			CPU 3	
index	1	2	3	4	5	6	7	8
Step 1	3	1	4	5	2	3	6	7
Step 2	3	4	**8**	5	7	**10**	6	**13**
Step 3			**8**			**18**		**31**
Step 4	3	4	8	13	15	18	24	31

- Analysis

  Step 1 : $O(\log n)$
  Step 2 : $O(\log n)$
  Step 3 : $O(\log (n/\log n)) = O(\log n)$
  Step 4 : $O(\log n)$
  Total work : $(n/\log n) * O(\log n) = O(n)$

  It is work optimal!