Accelerated Crowding

Processor Advantage

- Processor advantage of an algorithmic instance is the ratio of the number of processors available to the problem size

input of size : n

processors : p

processor advantage : p/n

Accelerated Crowding

- Suppose for a problem P we have a super-fast but wasteful-in-processors algorithm
- Say, we want to design a moderately fast, economic-in-processors algorithm for P
- Sometimes a solution can be found by iteratively reducing the problem size so that the processor advantage of the new instances increases exponentially through the iterations

Minimum on COMMON CRCW PRAM

$4 \quad 3 \quad 1 \quad 2 \longrightarrow 1$

n^2 processors.

$n^2 = 16$

$\forall \; \underline{i, j}$

$$\begin{array}{c|cccc|c} & 4 & 3 & 1 & 2 & \\ \hline 4 & 0 & 1 & 1 & 1 & 1 \\ 3 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \checkmark \\ 2 & 0 & 0 & 1 & 0 & 1 \end{array}$$

MINIMUM 1

$O(1)$ time
using n^2 processor
processor adv: $\frac{n^2}{n} = n$

MIN1

Input: Array $A[1 \dots n]$ of integers.

Output: The minimum integer R in A. Model: COMMON CRCW PRAM

Step 1: pardo for $1 \le i \le n, 1 \le j \le n$ $\quad B[i][j] = (A[i] > A[j])$;

Step 2: pardo for $1 \le i \le n$ $\quad C[i] =$OR-COMMON$(B[i])$;

Step 3: pardo for $1 \le i \le n$ $\quad$ if $(C[i] == 0)$ $\; R = A[i]$;

Step 4: return R;

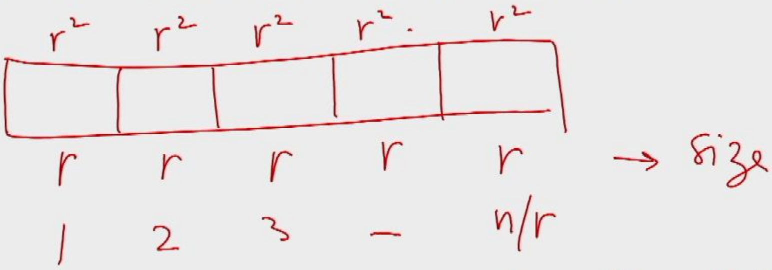
MINIMUM 1

- Exercise: Rewrite the above algorithm so that the processor allocation is obvious
- Hereafter, in algorithm descriptions the pardo variable will not always represent processor indices, but will still be indicative of the parallelism
- In each case, figure out how the processor allocation should go

MINIMUM 1

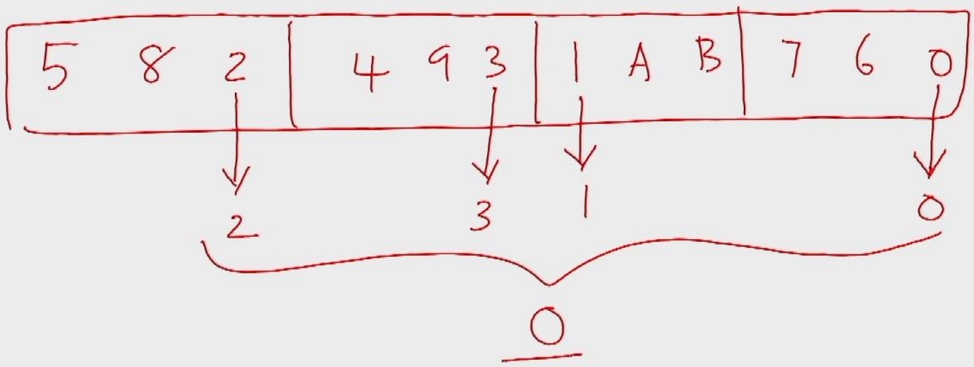
- Algorithm MINIMUM-1 runs in $O(1)$ time with n^2 processors
- Super-fast, but wasteful-in-processors
- Now, suppose we want to design an algorithm that uses nr processors, for $1 < r < n$
- That is, the initial processor advantage is r
- We divide the array into segments of size r each, allocate r^2 processors for each, solve each in $O(1)$ time; so, we are left with an instance of size n/r—the processor advantage is now r^2

Minimum on COMMON CRCW PRAM



processors: $r^2 \times \frac{n}{r} = \underline{\underline{nr}}$

elements $= r \times \frac{n}{r} = \underline{\underline{n}}$



MINIMUM 2

Input: Array $A[1 \dots n]$ of integers; r the processor advantage.
Output: The minimum integer R in A. Model: COMMON CRCW PRAM

Step 0: if $(n == r)$ return MIN1($A[1 \dots n]$);

Step 1: pardo for $1 \leq i \leq n/r$ $B[i]$ =MIN1($A[(i-1)r + 1 \dots ir]$);

Step 2: return MIN2($B[1 \dots n/r], r^2$);

problem size = n

processors = nr

pr. advantage = r

new problem size = $\underline{n/r}$

processors = nr

pr. advantage = $\frac{nr}{n/r} = r^2$

MINIMUM 2

$$T(n,r) = T\left(\frac{n}{r}, r^2\right) + c$$

$\frac{n/r}{r^2} = n/r^3$ segment

pr. adv = $\frac{nr}{n/r^3} = r^4$

If $T(n,r)$ is the time taken by a call MIN2($A[1\dots n], r$), then for some constant c,

$$T(n,r) = T(n/r, r^2) + c = T(n/r^3, r^4) + 2c = T(n/r^7, r^8) + 3c = \cdots = T(n/r^{2^s-1}, r^{2^s}) + sc$$

With $s = O(\log(\frac{\log n}{\log r}))$, thus $T(n,r) = O(s) = O(\log(\frac{\log n}{\log r}))$.

$$\log_r n + 1 = 2^{s+1}$$

$$\log_2 (\log_r n + 1) = s + 1$$

$$s = \log_2 (\log_r n + 1) - 1$$

$$= O(\log_2 \log_r n)$$

if $r = n^{\epsilon}$

i.e, $p = nr = n^{1+\epsilon}$

for a small const ϵ

$$\log r = \epsilon \log n$$

i.e, $\dfrac{\log n}{\log r} = \log_r n = \dfrac{1}{\epsilon}$

$$\log_2(\log_r n) = \log_2\left(\frac{1}{\epsilon}\right)$$

$$= O(1)$$

$O(1)$ time using $n^{1+\epsilon}$ processors

$O(\log \frac{1}{\epsilon})$

if $r == 2$

$O(\log_2 \log_2 n)$ time

$\#$ processors $= 2n$

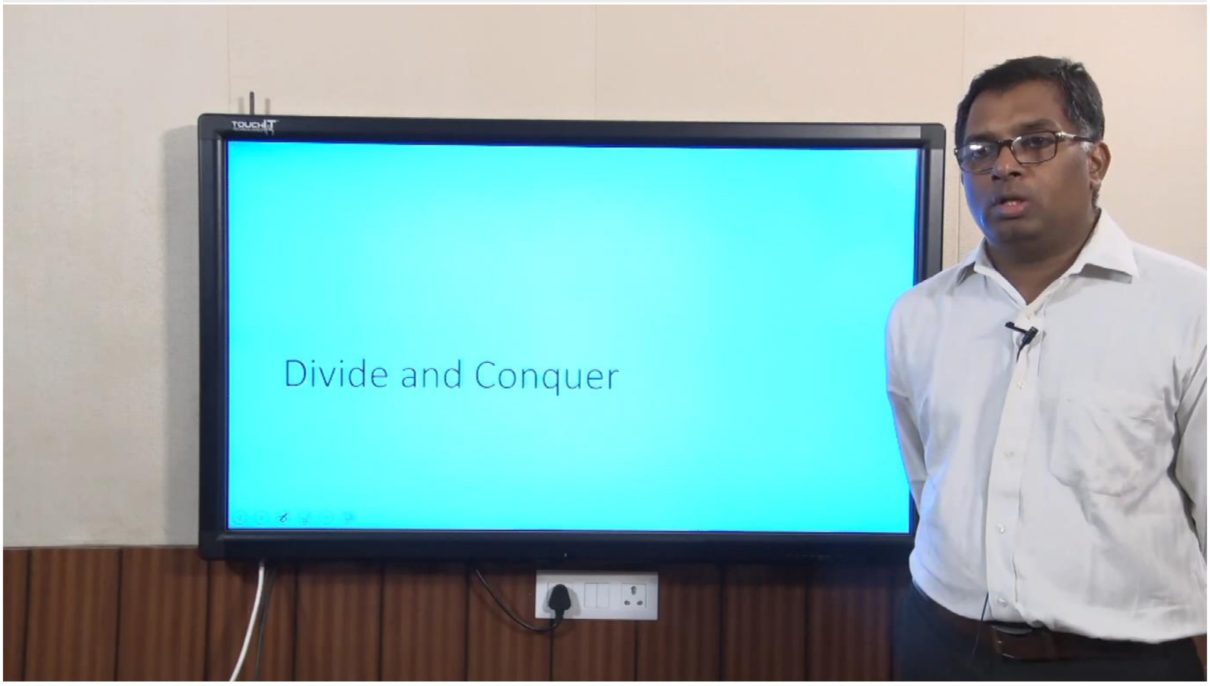
$\log\log(2^{65536}) = \log 65536 = 16$

$O(\log\log n)$ time n processors

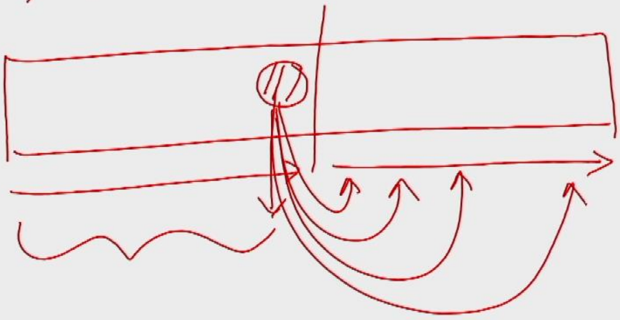
MINIMUM 2

For some constant ϵ, $0 < \epsilon < 1$, let $r = n^{\epsilon}$. Then $T(n, n^{\epsilon}) = O(1/\epsilon) = O(1)$. That is, **the minimum of n numbers can be found in $O(1)$ time with $n^{1+\epsilon}$ processors on a COMMON CRCW PRAM.**

Instead let $r = 2$. Then $T(n, 2) = O(\log\log n)$. That is, **the minimum of n numbers can be found in $O(\log\log n)$ time with n processors on a COMMON CRCW PRAM.**



Prefix Sums



4 8 1 5 2 7 3 6

4 8 1 5

4 8 1 5

4 8 1 5

4 12 1 6

4 12 13 18 2 9 12 18

4 12 13 18 20 27 30 36

Prefix Sums 2

PREFIX-SUMS-2

Input: Array $A[1 \ldots n]$ of integers. For simplicity, assume that n is a power of 2.

Output: An array $B[1 \ldots n]$ such that $B[i] = \Sigma_{j=1}^{i} A[j]$. Model: CREW PRAM.

```
{
    if (n == 1) return A;
    pardo for 0 ≤ i ≤ 1
        B[in/2 + 1 ... (i + 1)n/2] =PREFIX-SUMS-2(A[in/2 + 1 ... (i + 1)n/2]);
    pardo for 1 ≤ i ≤ n/2
        B[n/2 + i]+ = B[n/2];
    return B;
}
```

Prefix Sums 2

$$T(n) = T(n/2) + c_1$$
$$= T(n/4) + 2c_1$$
$$= T(n/8) + 3c_1$$
$$= T(n/2^k) + kc_1$$

Time : $T(n) = T(n/2) + c_1 = O(\log n)$. Cost: $C(n) = 2C(n/2) + c_2 n = O(n \log n)$.

$$= T(1) + c_1 \log n$$
$$= O(\log n)$$

$n/\log\log n$ processors are there

sequentially : $O(\log\log n)$

$n/\log\log n$ local minima

$$\log\log\left(\frac{n}{\log\log n}\right)$$

$$= \underline{O(\log\log n)}$$

cost = $O(n)$

COMMON CRCW PRAM