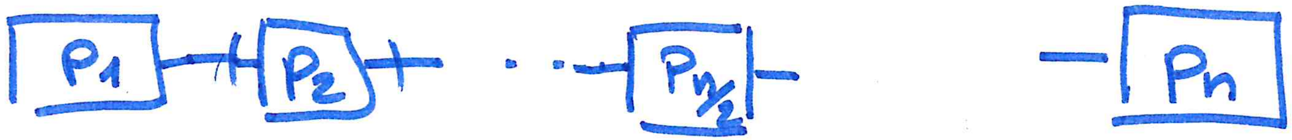
I architettura parallela a memoria distribuita

Array Lineari



Parametri di rete

$\gamma = 2$ ottimo per la realizzazione

$\delta = n-1$ lower bound per MAX $\delta \sim n$

$\beta = 1$ " " " ORDINAMENTO $\frac{n}{2}$

Ricordiamo che su PRAM abbiamo

MAX $p = \frac{n}{\log n}$ $T = \log n$

ORDINAM. $p = n$ $t = \log n$ (Cole)

IDEA: abbassare i processori

Problemi che affronteremo:
- shuffle
- MAX
- ORDINAMENTO

Array lineare

P_1 A[1] — P_2 A[2] — . . . — P_n A[n]

primitiva x shuffle

SWAP(k, k+1)

P_k — P_{k+1}

A[k+1]   A[k]

P_k	P_{k+1}
S →	← S
R ←	→ R
A[k]=A[k+1]	A[k+1]=A[k]

3 passi paralleli (tempo)

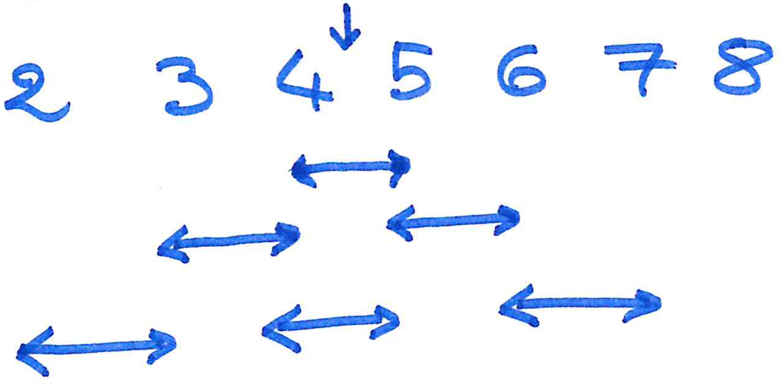
Problema shuffle

metà

Input: $A[1]\ A[2]\ \dots\ A[s]\ A[s+1]\ \dots\ A[2s]$

Output: $A[1]\ A[s+1]\ A[2]\ A[s+2]\ \dots\ A[s]\ A[2s]$

Soluzione per 8 processori



passi paralleli

1
2
3

idea:
è un albero
di swap contigui

Prestazione:

$\#\,proc = 2(s-1)$ tempo parallelo $= 3(s-1)$

sequenzialmente shuffle $T = \Theta(s^2)$

$$E \approx \frac{s^2}{s \cdot s} = c$$

OTTIMALE

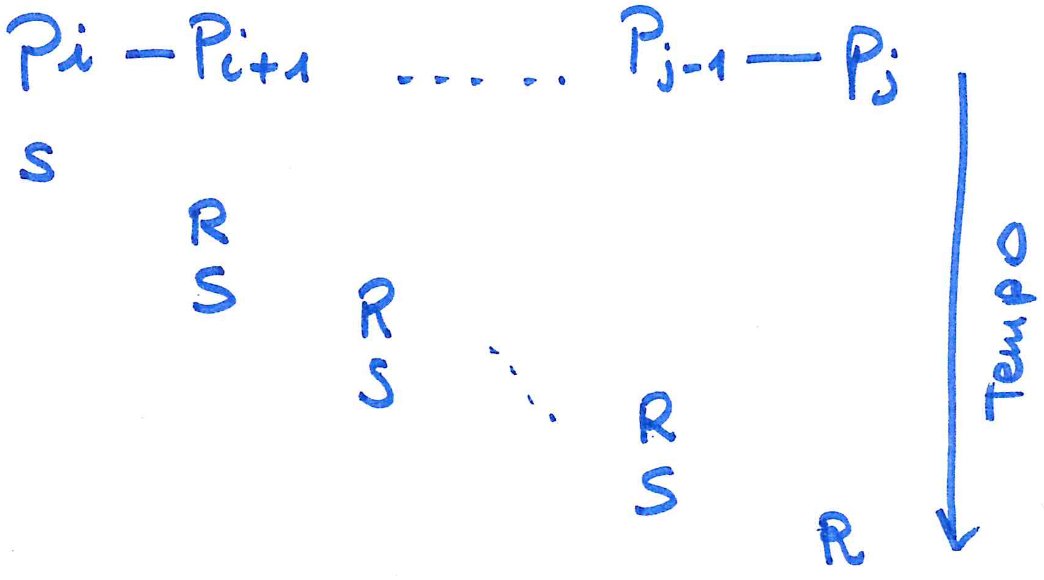
Primitiva per MAX

SEND (i , J)

$P_i \; \ldots\ldots \; P_j$

$A[i] \longrightarrow j$

$d(i,j)$ ← entra nel costo del problema



TOT: $2\, d(i,j) = 2\,|i-j|$ n° di passi paralleli

NOTA : la Trasmissione non è più costante come nelle PRAM _

Problema MAX

INPUT : $A[1]\ A[2] \ldots A[n]$

OUTPUT: Cont. $(P_n) = \max \{ A[i] \mid 1 \le i \le n \}$

Il Tempo per MAX su array lineari è limitato inferiormente da n

PRAM $\log n$

Sequenz. n

IDEA

1) Si considera l'algoritmo per SOMMATORIA delle PRAM

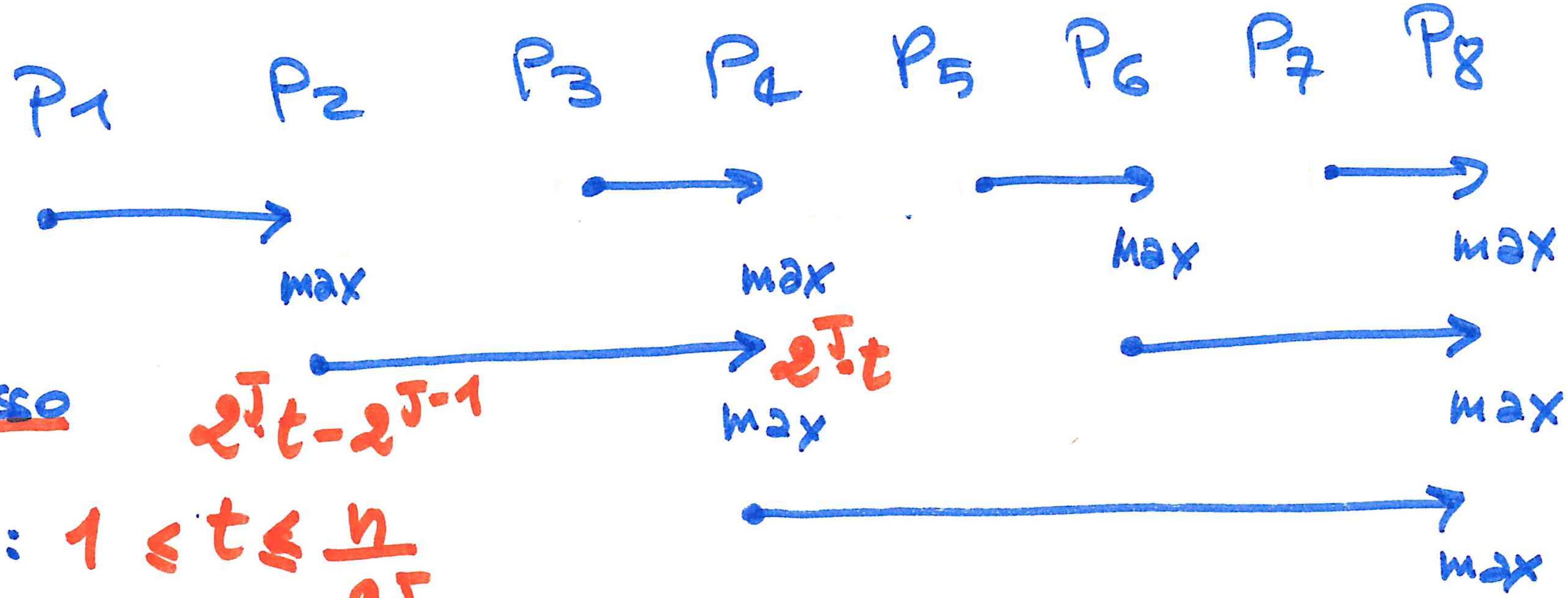
2) Riduzione dei processori per abbassare $\Omega(n)$ su array ad n processori

Opereremo in questo modo:

J-esimo passo
- confronto con numeri a distanza 2^{J-1}
- selezione del max
- memorizzato nel proc. di indice $2^J \cdot t$

Numero di passi è: $\log n$

Es:



J-esimo passo

dove: $1 \leq t \leq \frac{n}{2^J}$

(*) CODICE PER MAX

```
for J = 1 to log n
        for K ∈ { 2^J·t - 2^(J-1) | 1 ≤ t ≤ n/2^J } par do
(send)       SEND(K, K+2^(J-1))
        for K ∈ { 2^J t | 1 ≤ t ≤ n/2^J } par do
(compare)    if ( A[K] < A[K-2^(J-1)] ) then
                  A[K] = A[K-2^(J-1)]
```

Prestazioni per MAX del codice (*)

Tempo

(send) $\quad 2 \cdot 2^{J-1} \qquad J = 1, \dots, \log n$

(compare) $\quad 2 \qquad J = 1, \dots, \log n$

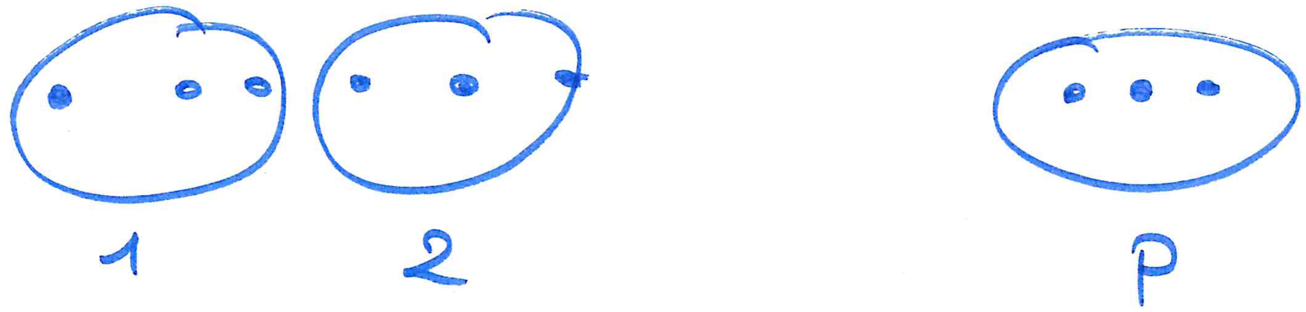
$$Tot = \sum_{J=1}^{\log n} 2 \cdot 2^{J-1} + \sum_{J=1}^{\log n} 2 = \sum_{j=1}^{\log n} 2^J + 2 \log n =$$

$$= 2^{\log n + 1} - 1 - 1 + 2 \log n = 2n - 2 + 2 \log n = O(n)$$

processori : n $\qquad E \to 0 \qquad$ non va bene

Suggerimento : riduciamo i processori da n a p
in questo modo operiamo sul parametro
δ = distanza max tra i processori

$n =$



- in questo modo ogni processore conterrà n/p dati

Nuovo algoritmo:

- Un processore seleziona il max sequenzialmentra tra i suoi n/p numeri
- si esegue il codice per MAX su p processori

Prestazioni

processori: p

Tempo: $O(n/p) + O(p)$

efficienza: numeratore $= n$

denominatore $= ?$

denominatore:

$$p\left(O(n/p) + O(p)\right) = \underbrace{O(n) + O(p^2)}_{O(n)}$$

Per avere $E \to C \neq 0$ scelgo $p^2 = n \Rightarrow p = \sqrt{n}$

Di conseguenza

MAX è risolto efficientemente su array lineari con

$p = \sqrt{n}$ e $T = O\left(\frac{n}{\sqrt{n}}\right) + O(\sqrt{n}) = O(\sqrt{n})$

Nuovo problema ORDINAMENTO

primitiva 1)

SWAP(i, j)

$\boxed{P_i}$ $\qquad$ $\boxed{P_j}$

$A[j] \longleftrightarrow A[i]$ $\quad$ devono fare

soluzione

I soluzione

Tempo

$4\, d(i,j) + 1$

SEND(i,j) $\qquad$ SEND(j,i)

$A[i] = A[j]$ $\qquad$ $A[j] = A[i]$

II soluzione: SEND simultanee

Abbiamo 2 casi: distanza tra i processori
- PARI
- DISPARI

I CASO: $d(i,j) = 2k+1$ dispari

$\Downarrow$

processori pari

P_i $\quad$ P_{k+i} $\quad$ P_{k+i+1} $\quad$ P_j

SEND$(i, k+i)$ $\quad$ SEND$(j, k+i+1)$

$S \longleftrightarrow S$

$R \longleftrightarrow R$

SEND$(k+i, i)$ $\quad$ SEND$(k+i+1, j)$

A $\quad$ A

II CASO: lasciato come esercizio

I CASO: Tempo $= \underbrace{2K}_{1^\circ \text{ SEND}} + 2 + \underbrace{2K+1}_{2^\circ \text{ SEND}} =$ ← Assegnamento

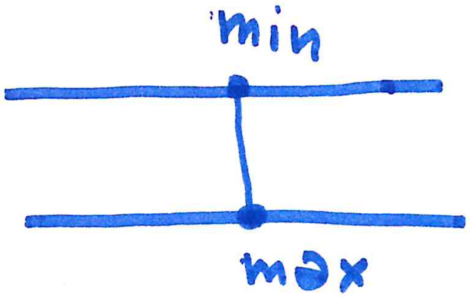
$$= 4K+3 = 2(2K+1)+1 = 2\,d(i,j)+1$$

II CASO: Tempo $= 2\,d(i,j)+3$

- ALTRA PRIMITIVA CHE SERVE PER L'ORDINAMENTO

MINMAX (K, K+1)

P_K	P_{K+1}
$\min\{A[k], A[k+1]\}$	$\max\{A[k], A[k+1]\}$



MINMAX realizza il confrontatore

MINMAX(K, K+1) P_k P_{k+1}

S S

R R

if (A[k] > A[k+1]) if (A[k+1] < A[k])

A[K] = A[k+1] A[k+1] = A[k]

Tempo: $2+2=4$ Tempo costante

Si può generalizzare a MINMAX(i, j)

si sfrutta l'algoritmo per SWAP(i, j) + TEST

Tempo: dispari pari

$2\,d(i,j)+2$ $2\,d(i,j)+4$