INTRODUZIONE
Lezione n°1

Algoritmi Avanzati

a.a.2011/2012

Prof.ssa Rossella Petreschi

Sistemi Concorrenti

Vogliamo analizzare come si deve progettare un algoritmo quando, per velocizzare la computazione, si vogliono usare contemporaneamente più entità addette al calcolo.

Parleremo di:

Computazione parallela e algoritmi paralleli

Computazione distribuita e algoritmi distribuiti

Differenza fra sistema concorrente e sistema sequenziale

Sistema concorrente	**Sistema sequenziale**
Le entità di un sistema concorrente non conoscono lo stato globale, ma conoscono solo il loro stato e quello delle entità adiacenti	Le decisioni prese si basano sullo stato globale del sistema (non è detto che l'accesso all'intero stato avvenga in una singola istruzione)
La relazione d'ordine temporale indotta negli eventi che costituiscono l'esecuzione di un algoritmo concorrente non è totale: due entità *p* e *q* possono richiedere di accedere contemporaneamente alla stessa risorsa *r*.	Gli eventi che costituiscono l'esecuzione di un algoritmo sequenziale sono totalmente ordinati: un processo *p* accede ad una risorsa *r* solo dopo che il processo *q* ha completato di usufruire della risorsa *r*.
Più computazioni sono possibili contemporaneamente	Dato un programma e un input solo una singola computazione é possibile

Classificazione di Flynn

SISD: Single Instruction, Single Data; è la tradizionale macchina sequenziale di **Von Neumann** in cui, ad ogni passo, l'unica unità di controllo esegue una sola istruzione che opera su di un singolo dato.

MISD: Multiple Instructions, Single Data; (si tratta di un'estensione della macchina sequenziale SISD) in cui ogni processore esegue un'operazione differente da quella svolta dagli altri su di un singolo dato comune a tutti i processori.

SIMD: Single Instruction, Multiple Data; macchina in cui ogni processore esegue la stessa istruzione su dati differenti; essa costituisce il modello base per la **PRAM (Parallel Random Access Machine)**.

MIMD: Multiple Instructions, Multiple Data; ogni processore ha un suo insieme di dati ed un suo insieme di istruzioni per elaborarli; i processori sono collegati attraverso reti di interconnessione **(Sistemi Distribuiti)**.

Computazione parallela

Cosa si vuole?

- velocizzare il tempo di esecuzione globale di un singolo “task”;
- aumentare il numero totale dei task eseguibili in un fissato intervallo di tempo.

Come si ottiene questo risultato?

- incrementando il parallelismo delle macchine.

Come si utilizza il parallelismo delle macchine?

- traducendo un programma sequenziale esistente;
- progettando un nuovo algoritmo basato su paradigmi e tecniche capaci di concettualizzare il parallelismo.

Macchina parallela

Sato di un computer:

lista del contenuto di tutte le celle di memoria.

Parallelismo di un computer:

massimo numero di celle che cambiano il loro stato durante l'esecuzione di un singolo ciclo operativo della macchina.

Computer letteralmente seriale:

valore del parallelismo uguale ad 1.

Parallelismo dei processori

Una macchina contenente un insieme di processori, tipicamente dello stesso tipo, interconnessi in modo tale da consentire il coordinamento delle attività e lo scambio dei dati viene detta parallela.

L'interconnessione è di tipo

- Memoria condivisa o
- Rete di Interconnessione

Parallel Random Access Machine
(Fortune-Wyllie 1978)

Il modello di computazione PRAM impiega p processori sincroni tutti aventi tempo di accesso unitario ad una memoria condivisa.

Ogni processore ha una propria memoria locale e tutti i processori lavorano in modo sincrono.

In ogni intervallo temporale unitario, un processore può:

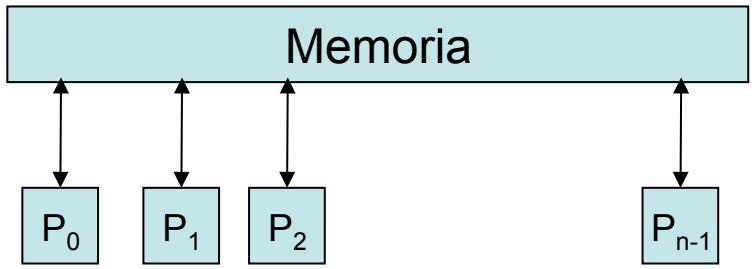
• leggere nella memoria condivisa (ovvero copiare il contenuto di una cella della memoria condivisa in uno dei registri della propria memoria locale);

• scrivere nella memoria condivisa (ovvero copiare il contenuto di uno dei registri della propria memoria locale in una cella della memoria condivisa);

• compiere una qualche computazione nella propria memoria locale.

Concorrenza in lettura e scrittura

PRAM



- **EREW**: Exclusive Read, Exclusive Write;
- **CREW**: Concurrent Read, Exclusive Write;
- **ERCW**: Exclusive Read, Concurrent Write;
- **CRCW**: Concurrent Read, Concurrent Write;

Scrittura Concorrente

Nei modelli a scrittura concorrente, quando più processori richiedono di scrivere in una stessa locazione di memoria condivisa, viene consentita la scrittura ad un solo processore in accordo con uno dei seguenti criteri di scelta:

- stabilita una scala di priorità tra i processori, si permette la scrittura solo al processore nella prima posizione della scala
  (CRCW prioritaria);
- supposto che tutti i processori vogliano scrivere lo stesso valore, consentire la scrittura ad uno qualunque dei processori
  (CRCW arbitraria).

Esistono in letteratura altri criteri, anche se meno usati, che permettono ad un fissato processore di scrivere nella cella di memoria il valore di una funzione di tutti i dati (ad esempio il massimo, lo xor, la somma).

E’ da notare che…

PRAM ignora aspetti realistici quali:

- **Sincronizzazione;**
- **Affidabilità;**
- **Velocità di comunicazione;**
- **Passaggio dei messaggi;**
- **Località dei dati.**

Complessità parallela

- **Tempo di esecuzione (T_p)**: si calcola come differenza tra il tempo in cui tutti i processori hanno terminato il loro lavoro e il tempo di inizio del primo processo.
- **Numero di processori (N_p)**: viene considerato il massimo numero di processori utilizzati.
- **Costo (C_p):** di un algoritmo è dato dal prodotto del tempo di esecuzione per il numero di processori.
- **Speed-up (Su):** è una misura utile a determinare quanto un algoritmo parallelo sia veloce rispetto al migliore algoritmo seriale conosciuto per lo stesso problema. Si calcola come rapporto tra il tempo impiegato nel caso pessimo dal miglior algoritmo seriale e il tempo del nostro algoritmo parallelo. $Su = T_s / T_p$.
- **Efficienza (Eff):** è il rapporto tra la complessità nel caso pessimo dell'algoritmo seriale ottimo e il costo dell'algoritmo parallelo. $Eff = T_s / C_p$.

Speed-up / efficienza

Lo **speed-up** è una misura utile a determinare quanto un algoritmo parallelo sia veloce rispetto al migliore algoritmo seriale conosciuto per lo stesso problema. Se si considera lo speed-up a partire da un algoritmo seriale ottimo, lo speed-up ideale è uguale al numero di processori, poiché un algoritmo che lavora su N_p processori può essere al più N_p volte più veloce del corrispondente algoritmo seriale. Quando si raggiunge lo speed-up ottimo si ottiene l'algoritmo parallelo ottimo (se lo era il seriale). Nel caso in cui lo speed-up sia maggiore di N_p, l'algoritmo seriale utilizzato non è ottimo e se ne può trovare uno migliore eseguendo serialmente i passi dell'algoritmo parallelo utilizzato.

L'**efficienza** deve risultare minore o uguale a 1. Se risultasse maggiore di 1 allora l'algoritmo seriale non sarebbe ottimo e sarebbe possibile trovarne uno migliore simulando l'algoritmo parallelo con l'esecuzione seriale degli stessi calcoli dell'algoritmo parallelo su di un unico processore.

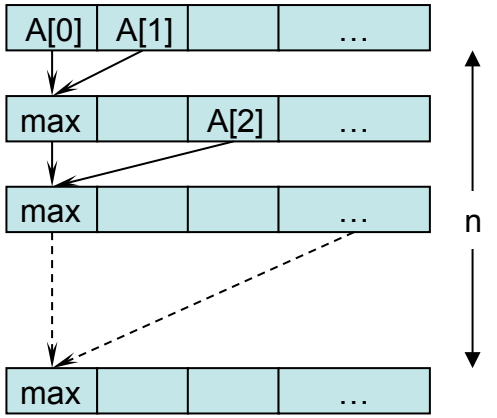
Bisogna ricordare che il numero N_p di processori risulta vincolato da considerazioni di carattere tecnologico ed economico (costo iniziale e di manutenzione); in pratica si cerca di rendere N_p indipendente dalla dimensione n del problema e comunque abbastanza piccolo. Volendo esprimere il rapporto esistente tra N_p ed n, possiamo utilizzare un'equazione del tipo:

$$N_p = f(n) = n^{1-x} \qquad \text{con } 0 \leq x \leq 1$$

Esempio di Algoritmo su P-RAM di tipo EREW

```
SeqMax(A, n)
begin
    for i = 1 to n-1 do
        if A[ i ] > A[ 0 ] then A[ 0 ] = A[ i ]
    return A[ 0 ]
end
```

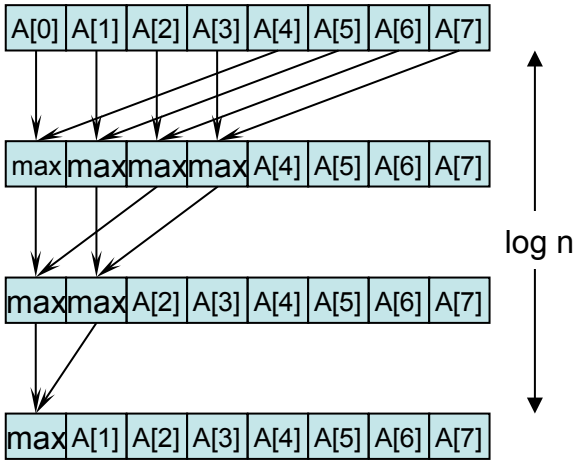
Lavora un solo processore



```
ParMax(A, n) begin
    for i = 1 to ⌈log n⌉ do
        k = ⌈n/2^i⌉
        for j = 0 to (k-1) pardo
Pj:         if A[ j ] < A[ j+k ] then A[ j ] = A[ j+k ]
    return A[ 0 ]
end
```

Si adoperano n processori

Tempo parallelo logaritmico



Esempio di Algoritmo su P-RAM di tipo CREW

Si adoperano n processori

Assumiamo A[i] tutti distinti

```
Cerca(A, n, x)
begin
    indice = -1
    for i = 0 to n-1 pardo
Pi:     if A[ i ] = x then indice = i     /* lettura concorrente di x */
    return indice
end
```

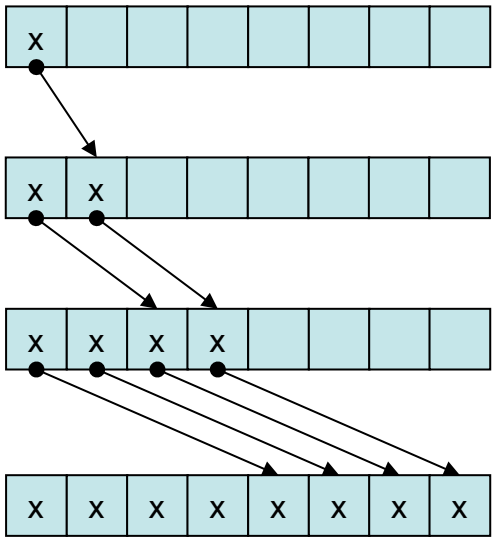
Tempo parallelo costante

Se A può contenere elementi ripetuti allora più processori possono scrivere concorrentemente la variabile indice: il modello necessario in tal caso è la P-RAM CRCW con scrittura concorrente basata su priorità dei processori (es viene scritto il valore dato dal processore di indice massimo tra quelli che tentano di scrivere).

Broadcast su P-RAM EREW

Si adoperano n processori

Broadcast(x)
begin
P_0: A[0] = x
    for i = 0 **to** ⌈log n⌉ -1 **do**
        for j = 2^i **to** 2^{i+1}-1 **pardo**
P_j: A[j] = A[j-2^i]
end



Tempo parallelo logaritmico

Uso di Broadcast su P-RAM

Si adoperano n processori

```
Cerca(A, n, x)
begin
    Broadcast(x)
    indice = -1
    for i = 0 to n-1 pardo
Pi:     if A[ i ] = x [ i ] then indice = i
    return indice
end
```

Tempo parallelo logaritmico

Se gli A[i] sono tutti distinti il modello è P-RAM EREW
altrimenti è ERCW per la scrittura dell'indice (scrittura concorrente basata su priorità dei processori)

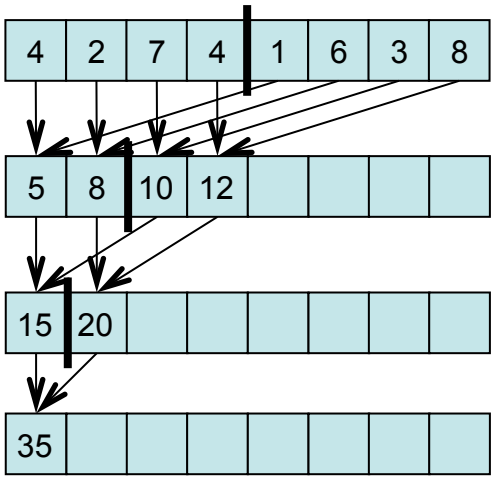
Somma su P-RAM EREW

Si adoperano n processori

Per semplicità assumiamo n potenza di 2

Somma(A)
begin
    for i = 1 **to** log n **do**
        for j = 0 **to** $n/2^i$ -1 **pardo**
P_j: A[j] = A[j] + A[j+$n/2^i$]
    return A[0]
end

Tempo parallelo logaritmico

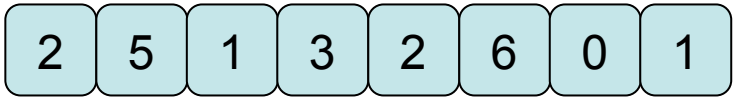


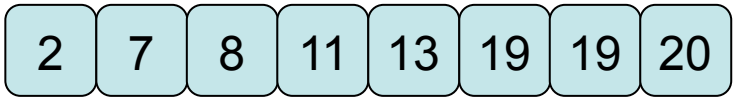
Tecnica della prima metà

Somme prefisse

Sequenziale

```
PrefixSum(A, n)
begin
    for i = 1 to n-1 do
        A[ i ] = A[ i ] + A[ i-1 ]
end
```





Tempo O(n)

Somme prefisse su P-RAM

P-RAM EREW con n processori

PrefixSum(A, n)
begin
    for i = 0 **to** ⌈log n⌉ -1 **do**
        for j = 0 **to** n-1 -2^i **pardo**
P_j: A[j+2^i] = A[j] + A[j+2^i]
end

Tempo Parallelo O(log n)