

## Operazioni Fondamentali su un Ipercubo

D'Amaro Michele 0521000061  
Valente Roberto 0520100439

Sistemi per l'elaborazione delle informazioni  
Anno Accademico 2004 - 2005

## Overview

- Data Broadcasting
- Window Broadcast
- Data Sum
  - SIMD Data Sum
  - SIMD All Sum
  - Prefix Sum
- Shift
  - SIMD Shift
  - MIMD Shift

## Alcune Definizioni.....

- Un'architettura multiprocessore è un computer parallelo che possiede due o più processori. Tali processori condividono una memoria comune o uno spazio degli indirizzi comuni (Quinn 1987).

## Alcune Definizioni.....

- Un ipercubo è una delle più efficienti architetture finora scoperte per il calcolo parallelo. Un ipercubo può simulare una qualsiasi architettura della stessa dimensione. In particolare un ipercubo di  $n$  nodi può simulare un array, un albero binario, o una mesh di alberi con  $O(n)$  nodi, solo con un piccolo fattore costante di ritardo
- Svantaggio: il numero di connessioni tra ogni processore cresce logaritmicamente con la dimensione della rete. E' evidente dunque che questo costituisce un problema per architetture dotate di molti processori

## Un po' di Notazioni

- $i^{(b)}$ : Indicheremo il numero che differisce da  $i$  esattamente nel bit  $b$
- $[]$ : Indicheremo un array
- $()$ : Indicheremo un processore
- $A[i]$ : Indicheremo l' $i$ -esimo elemento dell'array  $A$
- $A[j](i)$ : Indicheremo il  $j$ -esimo elemento dell'array  $A$  relativo al processore  $i$

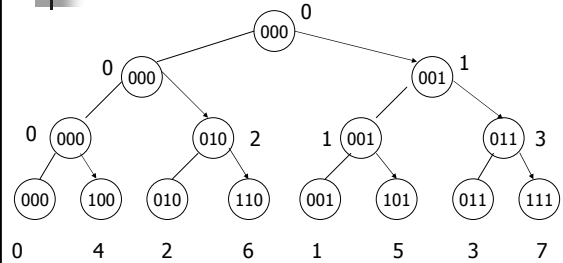
## Data broadcasting

- E' un meccanismo di trasmissione delle informazioni a tutti i processori dell'ipercubo.
- E' possibile realizzare la trasmissione delle informazioni attraverso un albero binario.

## Data broadcasting (2)

- Supponiamo di voler trasmettere il valore nel registro A, del processore 0 dell'ipercubo, al registro A di tutti gli altri processori rimanenti nell'ipercubo
- L'albero (nel nostro caso di altezza 3) sarà formato da nodi, che rappresentano i diversi processori, e archi di collegamento che diventeranno frecce quando si vorrà rappresentare un trasferimento di bit da un nodo all'altro

## Data Broadcasting (3): Albero di trasmissione



## Data broadcasting (4): Procedura 1

```
procedura Broadcast (A, d)
  for i=0 to d-1 do
    A(j(i)) ← A(j), (ji=0);
  end;
```

## Data broadcasting (5): Procedura 2

procedura Broadcast (A, d, k)

```
A(j):=null, (j≠k);
for i:=0 to d-1 do
  A(j(i)) ← A(j), (A(j) ≠ null);
end;
```

- k → processore k-esimo nel cui registro A c'è il valore da trasmettere al resto dei processori
- null → valore speciale altrimenti non ammissibile per A

## Data broadcasting (6): Complessità

- La complessità di tutte le procedure viste, relative al broadcasting, è  $O(d)$  dove d è la dimensione dell'ipercubo. In sostanza d coincide con il numero di iterazioni effettuate dal ciclo for.

## Window Broadcast

- È una tecnica di partizionamento di un ipercubo, di dimensione d, in finestre di taglia  $2^{(k)}$  processori ognuna
- Ogni finestra è un sotto ipercubo, ovviamente di dimensione  $2^{(k)}$
- Gli indici dei processori in ogni finestra differiscono nei loro k bit meno significativi
- Assumiamo che  $m_0, m_1, \dots, m_{(k-1)}$  siano i bit meno significativi del processore che conosce il bit da inviare

## Window Broadcast (2): Esempio

- Consideriamo il caso  $d=3$  e  $k=2$
- Supponiamo inoltre che i processori che conoscono il bit da inviare in broadcast siano quelli con i bit 0 e 1 uguali a 0

0	0	00	4	1	00
1	0	01	5	1	01
2	0	10	6	1	10
3	0	11	7	1	11

## Window Broadcast (3): Procedura 1

procedura WindowBroadcast (A,k)

```

for i:=0 to k-1 do
  A(j(i)) ← A(j), (ji = mi);
end;
```

## Window Broadcast (4): Procedura 2

- Anche per il Window Broadcast è previsto l'utilizzo del valore speciale null e il processore, che conosce il bit da inviare agli altri, è identificato attraverso un campo selected(j) posto a true dove j è l'indice di tale processore.

procedura WindowBroadcast (A, k)

```

A(j):=null, (selected(j) = false);
for i:=0 to k-1 do
  A(j(i)) ← A(j), (A(j) ≠ null);
end;
```

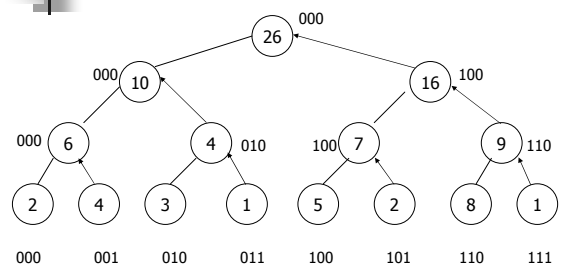
## Window Broadcast (5): Complessità

- La complessità delle procedure relative al window broadcasting è  $O(k)$  dove k è il numero dei bits meno significativi e coincide con il numero di iterazioni del ciclo for

## Data Sum

- L'operazione di Data Sum somma i valori dei registri A di tutti i processori appartenenti alla stessa finestra
- Il risultato è memorizzato nel registro A del processore che, nella finestra, ha indice minore

## Data Sum (2): Esempio SIMD Data Sum



## Data Sum (3): SIMD Data Sum Procedura

```

procedure SIMDDataSum (A,k)
  awake(j):=true;
  for i:=0 to k-1 do
  begin
    B(j(0)) ← A(j), ((j=1) and awake(j))
    awake(j):=false, (j=1);
    A(j):=A(j)+B(j), (awake(j));
  end;
end;

```

- awake(j) : Indica se il j-esimo processore è coinvolto nel processo di somma

## Data Sum (4): SIMD All Sum Procedura

- Se rimuoviamo le funzioni di selezione otteniamo una procedura che calcola la somma in ogni processore dell'ipercubo

```

procedure SIMDAllSum (A,k)
  for i:=0 to k-1 do
  begin
    B(j(0)) ← A(j);
    A(j):=A(j)+B(j);
  end;
end;

```

## Data Sum (5): Prefix Sum

- E' una tecnica utilizzata per il calcolo delle somme prefisse
- Supponiamo che un ipercubo di dimensione d sia partizionato in finestre di dimensione k
- Le somme prefisse nelle finestre di dimensione k possono essere facilmente calcolate se si conoscono i seguenti valori in ciascuna delle sottofinestre di dimensione k-1 che compongono la finestra di dimensione k:
  - Le somme prefisse nelle finestre di dimensione k-1
  - La somma di tutti i valori A nelle sottofinestre di taglia k-1

## Data Sum (6): Prefix Sum

- Le somme prefisse relative all'intera finestra sono ottenute come segue:
  - Se un processore è nella sottofinestra sinistra, allora la sua somma prefissa non cambia
  - Se un processore è nella sottofinestra destra, allora la sua somma prefissa è: la sua somma prefissa più i valori dei registri A nella sottofinestra sinistra

## Data Sum (7): Prefix Sum, esempio

- Le somme prefisse sono memorizzate nei registri S
- Le somme dei valori dei registri A dei singoli processori sono memorizzate nei registri T

Registri	0	1	2	3	4	5	6	7	Processori
A	2	4	3	1	5	2	8	1	
S	2	4	3	1	5	2	8	1	} Valori Iniziali per Finestre di Dim. 1
T	2	4	3	1	5	2	8	1	
S	2	6	3	4	5	7	8	9	} Valori per Finestre di Dim. 2
T	6	6	4	4	7	7	9	9	
S	2	6	9	10	5	7	15	16	} Valori per Finestre di Dim. 4
T	10	10	10	10	16	16	16	16	
S	2	6	9	10	15	17	25	26	} Valori per la Finestra di Dim. 8
T	26	26	26	26	26	26	26	26	

## Data Sum (8): Prefix Sum Procedura

```

procedure SIMDPrefixSum (A,k,S)
  begin
    S(i):=A(i);
    T(i):=A(i);
    for b:=0 to k-1 do
    begin
      B(i(b)) ← T(i);
      S(i):=S(i)+B(i)
      T(i):=T(i)+B(i);
    end;
  end;
end;

```

## Data Sum (9): Complessità

- Tutte le procedure viste relative al Data Sum hanno una complessità  $O(k)$ , corrispondente ai cicli di for svolti da ogni singola procedura.

## Shift

- L'operazione  $\text{shift}(A,i,w)$  shifta i dati contenuti nel registro A dei processori di "i" posizioni in avanti nella finestra di dimensione w dove w è una potenza di 2.
- In un ipercubo SIMD di dimensioni  $p=2^d$  i processori sono ordinati ciclicamente in  $0,1,2,\dots,p-1,0$

## SIMD Shift

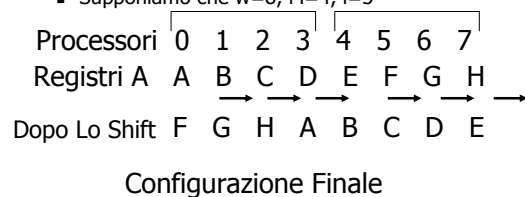
- La strategia è quella di ridurre lo shift, in una finestra di dimensione w, in due shift indipendenti in finestre di dimensione  $w/2$
- Usando ripetutamente questa strategia finiremo ad eseguire lo shift su finestre di dimensione 1 cioè shift nulli

## SIMD Shift (2)

- Supponiamo che la nostra finestra abbia dimensione 2M. Consideriamo 2 casi:
  - 1.  $0 \leq i < M$
  - 2.  $M \leq i < 2M$
- Se "i" non è in questo intervallo, può essere rimpiazzato da  $i \bmod 2M$  e ricondotto ad uno dei due casi

## SIMD Shift (3): Esempio

- Caso I
- Supponiamo che  $w=8, M=4, i=3$



## SIMD Shift (4)

- Gli otto processori possono essere partizionati in due finestre di dimensione quattro
- La sottofinestra  $s_x$  è costituita dai processori 0 - 3, mentre la sottofinestra  $d_x$  è costituita dai processori 4 - 7
- Dalla configurazione iniziale e finale notiamo che B, C, D sono inizialmente nella sottofinestra  $s_x$  e dopo lo shift risultano in quella  $d_x$
- Viceversa per i processori F, G, H

## SIMD Shift (5)

- Se scambiamo i processori B, C, D con F, G, H rispettivamente e poi eseguiamo uno shift di tre posizioni in ciascuna finestra otteniamo quanto segue:

Processori	0	1	2	3	4	5	6	7
Registri A	A	B	C	D	E	F	G	H
Scambio	A F G H			E B C D				
<b>Dopo lo Shift</b>	F	G	H	A	B	C	D	E

## SIMD Shift (6)

- In definitiva per eseguire uno shift di "i" posizioni, con  $0 \leq i < M$  in una finestra di dimensioni  $w=2M$  consideriamo le due sue sottofinestre di dimensione M ciascuna
- Dopo lo shift alcuni elementi, inizialmente nella sottofinestra  $s_x$ , si trovano nella sottofinestra  $d_x$  e viceversa (p.s. lo stesso numero di elementi)

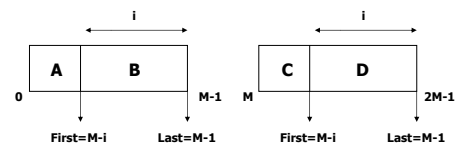
## SIMD Shift (7)

- Indichiamo con A e C, rispettivamente, gli elementi delle sottofinestre  $s_x$  e  $d_x$  che non cambiano sottofinestra dopo lo shift
- Indichiamo con B e D invece i rimanenti elementi delle sottofinestre  $s_x$  e  $d_x$

Processori	0	1	2	3	4	5	6	7
Registri A	A	B	C	D	E	F	G	H
Scambio	A	F	G	H	E	B	C	D
<b>Dopo lo Shift</b>	F	G	H	A	B	C	D	E

## SIMD Shift (8)

- Sia B che D contengono "i" elementi ed entrambi consistono di elementi che iniziano nella posizione  $first=M-i$  e finiscono nella posizione  $last=M-1$



## SIMD Shift (9)

- Eseguiamo ora lo scambio tra B e D



- Infine eseguiamo uno shift di i mod M posizioni in ciascuna sottofinestra di dimensione M



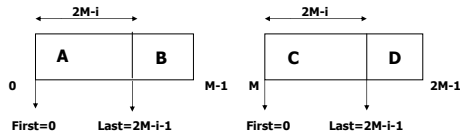
## SIMD Shift (10)

- II Caso ( $i > M$ )
- Consideriamo l'esempio precedente con  $i=6$

Processori	0	1	2	3	4	5	6	7
Registri A	A	B	C	D	E	F	G	H
<b>Dopo lo Shift</b>	C	D	E	F	G	H	A	B

## SIMD Shift (11)

- Consideriamo gli elementi nelle due sottofinestre della finestra di dimensione  $w$ . Osserviamo che A,B,E,F cambiano finestra tra la configurazione iniziale e finale.
- Indichiamo con B e D, rispettivamente, gli elementi delle sottofinestre  $s_x$  e  $d_x$  che non cambiano sottofinestra dopo lo shift
- Indichiamo con A e C invece i rimanenti elementi delle sottofinestre  $s_x$  e  $d_x$



## SIMD Shift (11)

- Dalla figura precedente possiamo vedere come sia A che C contengono  $2M-i$  elementi ed entrambi consistono degli elementi che cominciano con  $first=0$  e terminano con  $last=2M-i-1$
- Scambiamo A e C



## SIMD Shift (12)

- Shiftiamo di  $i \bmod M$  in ciascuna sottofinestra. ( $i \bmod M = 6 \bmod 8 = 6$ )



## SIMD Shift (12): Procedura

```

Procedure SIMDSHIFT (A,I,W)
  i:=i mod w;
  b:= RightmostOne(i);
  M:=w;
  k:=log2 M;
  for j:=k-1 downto 0 do
  begin
    M:=M div 2;
    if i <= M then begin
      first:=M-i;
      last:=M-1;
    end
    else begin
      first:=0;
      last:=2M-i-1;
    end
    A:=p mod M
    A(p(j))<- A(p), (first <= a <= last)
    i:=i mod M
  end
end
  
```

## SIMD Shift (13):

- "p" è l'indice dei processori
- "a" è la posizione nella finestra di dimensione M
- "RightmostOne(i)" è una funzione che restituisce la posizione del LSB di "i" che è uguale ad 1 (nel caso in cui  $i=0$  la funzione restituisce "-1")
  - Esempio:  $RightmostOne(12)=2$  perché 1100 è la rappresentazione binaria di 12 e il bit pari a 1 meno significativo è in posizione 2.
- La complessità dell'algoritmo visto è  $O(\log w)$   $O(\log w \log i)$  se  $w$  è una potenza di 2

## SIMD Shift (14):

- Applichiamo l'algoritmo ad un esempio
- Abbiamo un ipercubo di  $2^3=8$  processori numerati da 000 a 111 e vogliamo eseguir uno shift di  $i=3$  posizioni

Processori 000 001 010 011 100 101 110 111  
 Registri A A B C D E F G H

## SIMD Shift (15):

- Inizialmente:
  - $i=3$
  - $b=0$  (perché  $3=011$  in binario)
  - $M=8$
  - $k=3$
- Entriamo nel ciclo for:
  - $j=2$
  - $M=4$
  - $first=1$
  - $last=3$  (poiché  $(i=3) \leq (M=4)$ )

## SIMD Shift (16):

- Processori 000 001 010 011 100 101 110 111
- I processori 001, 010, 011 e 101, 110, 111 che hanno induce compreso tra first e last scambiano i valori dei loro registri ( $001 \leftrightarrow 101$ ,  $010 \leftrightarrow 110$ ,  $011 \leftrightarrow 111$ )
  - La situazione diventa la seguente con  $i=3$ :

A	F	G	H	E	B	C	D
0		0		0		0	

## SIMD Shift (17):

- Rientriamo nel ciclo for:
  - $j=1$
  - $M=2$
  - $first=0$
  - $last=0$  (poiché  $(i=3) > (M=2)$ )
- Tutti i processori di indice "0" scambiano il valore del loro registro A con quello del processore che differisce da loro nel secondo bit ( $000 \leftrightarrow 010$ ,  $100 \leftrightarrow 110$ )

G	F	A	H	C	B	E	D
0		0		0		0	

## SIMD Shift (17)

- Eseguiamo l'ultima iterazione del for:
  - $j=0$
  - $M=1$
  - $first=0$
  - $last=0$  (poiché  $(i=1) \leq (M=1)$ )
- Consideriamo sottofinestre di dimensione 1
- Tutti i processori che differiscono nel LSB scambiano i valori dei loro registri ( $000 \leftrightarrow 001$ ,  $010 \leftrightarrow 011$ ,  $100 \leftrightarrow 101$ ,  $110 \leftrightarrow 111$ )

F	G	H	A	B	C	D	E
---	---	---	---	---	---	---	---

## MIMD Shift

- Su un ipercubo MIMD uno shift di "i" posizioni è ottenuto eseguendo una serie di shift ognuno dei quali è una potenza di due
- Supponiamo di voler eseguire uno shift di "i" posizioni in una finestra di dimensione w con "i" e "w" entrambe potenze di due e con  $i < w$
- Ogni finestra di dimensione w è composta di un certo numero di sottofinestre di dimensione i

## MIMD Shift (2)

- Se numeriamo i processori secondo il codice di Gray otterremo che gli indici dei processori in ogni sottofinestra differiscono da quelli nella sottofinestra adiacente in esattamente un bit
- Inoltre se cambiamo questi bit gli indici in una sottofinestra sono in ordine inverso rispetto a quelli nella finestra adiacente

## MIMD Shift (3): Esempio

- Consideriamo un ipercubo con 8 processori i cui indici sono in codice Gray  
000 001 011 010 110 111 101 100
- Dividiamo la finestra in quattro finestre di dimensione 2 e otteniamo le seguenti coppie:  
{000,001} {011,010} {110,111} {101,100}
- Le prime due coppie differiscono nel bit centrale. Cambiando questo bit nella prima coppia otteniamo {010,011} che è l'ordine inverso della seconda coppia. La seconda e la terza coppia differiscono nel MSB. Se cambiamo il MSB nella seconda coppia otteniamo la coppia {111,110} che è l'ordine inverso della seconda coppia e così via.

## MIMD Shift (4)

- L'algoritmo consiste di due passi:
  - Passo 1: Ciascun processore in una sottofinestra di dimensione "i" trasmette il suo dato al processore corrispondente nella sottofinestra adiacente alla sua dx
  - Passo 2: Se  $i > 1$  allora in ciascuna sottofinestra di dimensione "i" i dati sono invertiti scambiandoli tra le sottofinestre di dimensione  $i/2$

## MIMD Shift (5): Esempio

- Consideriamo lo shift di  $i=4$  posizioni in un ipercubo di dimensione 8

Processori	000	001	011	010	110	111	101	100
Registri A	A	B	C	D	E	F	G	H
Passo 1	H	G	F	E	D	C	B	A
Passo 2	E F G H A B C D							

## MIMD Shift (6)

- Inizialmente dividiamo la finestra in due sottofinestre di dimensione quattro. Ciascun processore nella prima finestra trasmette il suo dato al processore nella seconda finestra che differisce da lui nel MSB.
- Ora i dati sono nelle giuste sottofinestre di taglia 4 ma in ordine inverso. Per ottenere l'ordine giusto i dati vengono scambiati tra coppie di processori in finestre di taglia 2.

000	001	011	010	110	111	101	100
A	B	C	D	E	F	G	H
H	G	F	E	D	C	B	A

## MIMD Shift (6)

- Quindi i processori nelle coppie ({000, 001}, {011, 010}) come quelli nelle coppie ({110,111},{101, 100}) scambiano i dati attraverso il bit in posizione 1.

000	001	011	010	110	111	101	100
A	B	C	D	E	F	G	H
H	G	F	E	D	C	B	A
→				→			
E	F	G	H	A	B	C	D

## MIMD Shift (7): Esempio

- Consideriamo uno shift di  $i=2$  posizioni in un ipercubo di otto processori.

Processori	000	001	011	010	110	111	101	100
Registri A	A	B	C	D	E	F	G	H
Passo 1	H	G	B	A	D	C	F	E
Passo 2	G H A B C D E F							

## MIMD Shift (8)

- Ogni sottofinestra di dimensione 2 invia i suoi dati alla sottofinestra adiacente alla sua dx
- La sottofinestra {000,001} trasmette a {011,010} attraverso il bit 1;
- {011,010} trasmette a {110,111} attraverso il bit 2;
- {110,111} trasmette a {101,100} attraverso il bit 1
- Infine {101, 100} trasmette a {000, 001} attraverso il bit 2

H G B A D C F E

- A questo punto il passo due che ci permette di scambiare i valori nei registri di sottofinestre  $i/2$  cioè "1" ottenendo

G H A B C D E F

## MIMD Shift (9) (complessità)

- Quando "i" è una potenza di 2 lo shift può essere fatto in tempo costante  $O(1)$
- Quando "i" non è una potenza di 2 uno shift di "i" posizioni richiede  $O(\log_2 w)$  dal momento che "i" può avere al massimo  $\log_2 w$  uni nella sua rappresentazione binaria