Algoritmi di trasformazione delle immagini con ipercubo

Corso di:
Sistemi per Elaborazione dell'Informazione
Prof. Carpentieri Bruno

Amato Aniello
Ingenito Antonio
A.A. 2004/05

Indice argomenti :

- Reti di confrontatori
- Principio zero-uno
- Rete di ordinamento bitonico
- Bitonic-sort
- Introduzione a operazioni di restringimento ed estensione delle immagini

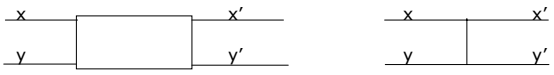
Reti di confrontatori

- Confrontatori e filo elettrico
- Definizione di una rete di ordinamento
- Analisi tempo di esecuzione di una rete di confrontatori
- Rete di confrontatori vs Macchina RAM

Confrontatori

Una rete di confrontatori è costituita eclusivamente da fili elettrici e confrontatori. Un confrontatore è un dispositivo che ordina 2 valori presi in input stabilendo tra questi il minimo e il massimo:

$x' = \min(x,y)$ $y' = \max(x,y)$



Il tempo di esecuzione di un confrontatore è O(1), inoltre ogni confrontatore produce i suoi valori di output solo quando entrambi i valori di input sono disponibili.

Filo elettrico

I fili elettrici possono connettere l'output di un confrontatore all'input un altro; in caso contrario essi sono fili di input o di output della rete.
Definiamo sequenza di input l'insieme dei valori da ordinare, tali valori entrano nella rete attraverso i fili di input; invece definiamo con il termine sequenza di output l'insieme dei valori identificati dai fili di output.
Un filo di input ha profondità 0, mentre un filo di output di un confrontatore che in input ha 2 fili le cui profondità sono rispettivamente d_x e d_y ha profondità $\max(d_x,d_y)+1$.

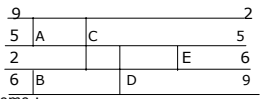
Interconnessione di confrontatori

Il grafo di interconnessione dei confrontatori deve essere aciclico: se si segue un cammino dall'output di un dato confrontatore all'input di un altro , all'output e poi all'input e così via , il cammino seguito non deve mai tornare indietro formando un ciclo su se stesso né passare attraverso uno stesso confrontatore due volte.

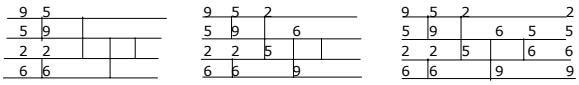
Una rete di confrontatori è come una procedura dal momento che specifica come devono essere eseguiti i confronti, ma ne defferisce perchè il numero di confrontatori in essa contenuti dipende dai fili di input e di output.

Un esempio……

Supponiamo che la sequenza <9,5,2,6> appaia sui fili di input al tempo 0, al tempo 3 la sequenza sarà ordinata :



Vediamo come :



E' anche una rete di ordinamento .

Analisi tempo di esecuzione di una rete di confrontatori

Il tempo di esecuzione di una rete di confrontatori è il tempo richiesto perché tutti i fili di output ricevano i loro valori dopo che i fili di input abbiano ricevuto i loro, informalmente rappresenta il più grande numero di confrontatori che un elemento possa attraversare viaggiando da un filo di input ad un filo di output.

Reti di confrontatori vs Macchina RAM

A differenza della Macchina Ram una rete di confrontatori può eseguire solo operazioni di confronto(punto a sfavore)ma tali confronti possono essere eseguite in parallelo(punto a favore). La Macchina Ram non limita le sue operazioni a semplici confronti ma la sequenza di operazioni deve essere seriale.

Conseguenze :
1) Su una rete di confrontatori non possono essere eseguiti tutti i tipi di algoritmi(counting-sort : no) sulla macchina ram questo problema non esiste.
2) Su una rete di confrontatori l'ordinamento può essere eseguito in tempo sub-lineare , su una macchina ram no .

Principio zero-uno

Ci serviremo di questo principio per facilitare l'analisi della correttezza delle reti di confrontatori.
Il principio zero-uno afferma che se una rete di confrontatori ordina correttamente tutte le sequenze di zero e uno questa ordinerà correttamente numeri arbitrari di input (reali , interi e generalmente qualsiasi insieme di valori presi da un qualunque insieme totalmente ordinato) .

Per dimostrare il principio zero-uno ci serviremo del seguente lemma:
Se una rete di confrontatori trasforma la sequenza di input a=<a1 , a2 , ….. an> in b=<b1 , b2 , …. bn>allora per qualunque f crescente la rete trasforma f(a)=<f(a1), f(a2), … f(an)> in f(b)=<f(b1), f(b2), … f(bn)>

Dimostrazione del lemma

Se f è monotona crescente → un confrontatore singolo con input f(x) , f(y) produce in output f(min(x,y)) e f(max(x,y)) .

Procediamo per induzione:
Se 1 confrontatore prende in input x e y in output darà min(x,y) e max(x,y), applicando la f a x e y l'input del confrontatore sarà f(x), f(y) e l'output min(f(x),f(y)) e max(f(x),f(y)).

Adesso essendo f monotona crescente se x<=y anche f(x)<=f(y) di conseguenza avremo :
min(f(x),f(y)) = f(min(x,y)) e max(f(x),f(y)) = f(max(x,y))
Siccome il confrontatore produce i valori f(min(x,y)) e f(max(x,y)) con input f(x) , f(y) il lemma è dimostrato .

Dimostrazione del teorema

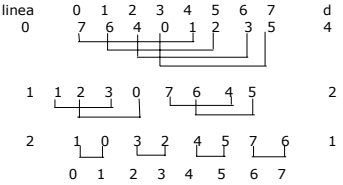
DIM:
Supponiamo per assurdo che la rete ordini tutte la sequenze di zero e uno, ma che esista una sequenza di numeri arbitrari che la rete non ordina correttamente e che quindi si verifichi tale situazione : $a_i < a_j$ ma la rete pone aj prima di ai.
Definiamo f una funzione monotona crescente
tale che :

$f(x) = 0 \leftrightarrow x <= a_j$ e $f(x) = 1 \leftrightarrow x > a_j$

Siccome la rete pone a_i prima di a_i per il lemma che abbiamo dimostrato prima segue che la stessa rete mette prima $f(a_j)$ di $f(a_i)$, ma poiché $f(a_j) = 1$ e $f(a_i) = 0$ si ottiene la contraddizione in quanto la rete non ordina correttamente la sequenza di zero-uno.

Una rete di ordinamento bitonico

Prima di definire una rete di ordinamento bitonico definiamo la sequenza bitonica che è una sequenza che prima cresce monotonamente e poi decresce monotonamente. Una rete di ordinamento bitonico non è altro che una rete che ordina sequenze bitoniche e lo fa confrontando gli input i e i+n/2 per ogni compreso tra 1 e n/2.
Esempio :



Bitonic-sort

```
Procedure bitonicSort(n)
   begin
        d = n/2
        while d > 0
           confronta e scambia gli elementi a distanza d
            d = d/2
        end
   end
```


Image trasformations

Per il restringimento , l'estensione e la rotazione in scala di un immagine sono stati sviluppati algoritmi efficienti , ma essendo tali
operazioni molto complesse dal punto di vista asintotico si tratta di algoritmi paralleli.
Assumiamo che N sia potenza di 2 e che N processori siano liberi, questi possono essere visti come una matrice NxN di cui gli indici della riga maggiore saranno usati per l'ipercubo SIMD mentre il codice grigio per l'ipercubo MIMD.
Il pixel I[i,j] dell'immagine sarà mappato con il processore in posizione (i,j) dell'ipercubo.

Restrizione ed espansione (1)

Il pixel confinante con il punto(i,j) è definito dal seguente insieme :
nbd(i,j) = { [u , v] | 0<=u,v < N , max { [u- i] , [v - i]} <=1}

La restrizione di I è definita da :

$$S^{q}[i,j] = \min_{[u,v]\in nbd\,(i,j)} \{ I[u,v] \}, q = 1,\ 0 \le i < N$$

$$S^{q}[i,j] = \min_{[u,v]\in nbd\,(i,j)} \{ S^{q-1}[u,v] \}, q > 1,\ 0 \le i,j < N$$

Restrizione ed espansione (2)

- mentre l'estensione da :

$$E^{q}[i,j] = \max_{[u,v]\in nbd\,(i,j)} \{ I[u,v] \}, q = 1,\ 0 \le i < N$$

$$E^{q}[i,j] = \max_{[u,v]\in nbd\,(i,j)} \{ E^{q-1}[u,v] \}, q > 1,\ 0 \le i,j < N$$

Sommario:

- MIMD Shrinking
- SIMD Shrinking
- Traslazione
- Rotazione
- Scaling

MIMD Shrinking (1)

- Sull'ipercubo MIMD R^q con $q=2^k$, R è calcolato in due fasi ottenuto dalla decomposizione della seguente formula:

$$R^q[i,j] = \min_{[i,v]\in B_{2q+1}(i,j)} \{I[i,v]\}, 0 \le i < N, 0 \le j < N \quad (1)$$

MIMD Shrinking (2)

- In:

$$left^q[i,j] = \min_{\substack{[i,v]\in B_{2q+1}[i,j] \\ v\le i}} \{I[i,v]\}, 0 \le i < N, 0 \le j < N \quad (2)$$

- E:

$$right^q[i,j] = \min_{\substack{[i,v]\in B_{2q+1}[i,j] \\ v\ge i}} \{I[i,v]\}, 0 \le i < N, 0 \le j < N \quad (3)$$

MIMD Shrinking (3)

- Avremo che la (1) sarà:

$$R^q[i,j] = \min\{left^q[i,j], right^q[i,j]\}, 0 \le i < N, 0 \le j < N \quad (4)$$

Dopo il primo ciclo di for con a=i, *left*(p) è il valore del pixel più piccolo dei 2^a processori a sinistra e questo è proprio nel registro I con $0 \le a < k$

MIMD Shrinking (4)

- Per completare l'esecuzione di *left*(p) bisogna considerare i valori dei 2^k pixel a sinistra sulla riga dell'immagine.
- Questo è giusto per uno shift a destra di 2^k.
- Lo shift è fatto per righe con un assegnazione di un numero molto grande (∞ per convenzione).
- Lo stesso è fatto per right.

Algoritmo (1)

```
procedure MIMDShrink;
    {Compute Rq for q = 2k on an MIMD hypercube}

begin
    {compute min of the left 2k pixels on the same row}
    {MIMDEShift does an ∞ fill instead of a 0 fill}
    left (p) := I (p);

    for i :=0 to k - 1 do
        begin
            C (p) := left (p);
            MIMDEShift (C, 2i, N);
            left (p) := min {left (p), C (p)};
        end
```


Algoritmo (2)

```
    C (p) := I (p);
    MIMDEShift (C, 2k, N);
    left (p) := min {left (p), C (p)};
    {compute min of the right 2k pixels on the same row}
    right (p) := I (p);
        for i :=0 to k - 1 do
            begin
                C (p) := right (p);
                MIMDEShift (C,-2i, N);
                right (p) := min {right (p), C (p)};
            end
    C (p) := I (p);
    MIMDEShift (C, -2k, N);
    right (p) := min {right (p), C (p)};
    R (p) := min {left (p), right (p)}
end;
```

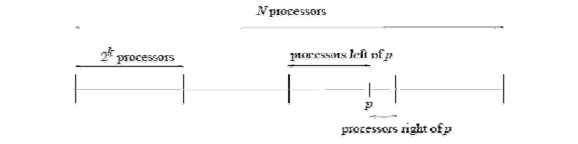
SIMD Shrinking (1)

- Quando lo shift è di 2^i in una finestra è di taglia N l'algoritmo impiega $O\left(\log\left(\frac{N}{2^i}\right)\right)$ in un ipercubo SIMD, con un semplice adattamento all'algoritmo precedente la complessità sarà O(k log N).
- Dobbiamo fare meglio

SIMD Shrinking (2)

- Un'immagine NxN è mappata in un ipercubo NxN
- R^q con $q=2^k$ può essere calcolato considerando gli N processori che rappresentano le righe dell'immagine e formato da molti blocchi
- Ogni blocco ha dimensione 2^k

SIMD Shrinking (3)



SIMD Shrinking (4)

- Ogni processore p esegue:
- left(p)=Il valore più piccolo dei pixel a sinistra di p ma con tutto il blocco 2^k.
- right(p)= Il valore più piccolo dei pixel a destra di p ma con tutto il blocco 2^k

SIMD Shrinking (5)

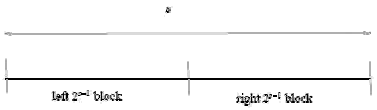
- Ora la matrice $R^q(p)$ contiene il minimo di:
- I(p);
- left(p)
- right(p)
- left(p+q) a condizione che p+q *è* in tutte le righe
- right(p-q) a condizione che p-q è in tutte le righe

SIMD Shrinking (6)

- Questo è vero se q non è potenza di 2
- E se usiamo $k = \lfloor \log_2 q \rfloor$ per definizione di left e right.
- *left (p)* e *right(p)* per 2^k blocchi possono essere eseguiti alla prima computazione, per 2^0 blocchi, 2^1 blocchi e così via.

SIMD Shrinking (7)

- Sia *whole*(*p*) il minimo di tutti i pixel nel blocco corrente contente p elementi di elaborazione (PE).
- Per 2^0 blocchi abbiamo *left*(*p*)=*right*(*p*)=∞ e *whole*(*p*) = *I*(*p*)



SIMD Shrinking (8)

- Ogni 2^s blocchi per *s*>0 consiste di due 2^{s-1} blocchi uno è il blocco a sinistra l'altro a destra. Gli elementi di elaborazione a sinistra ha *s*-1-mo bit =0 finché l'elemento *s*-1-mo di elaborazione a destra sia 1.
- Quindi usiamo un superscript per denotare la dimensione del blocco

SIMD Shrinking (9)

- Denotiamo *left* (*p*) con $left^s(p)$ quando il blocco ha dimensioni 2^s. Possiamo vedere che *p* è a sinistra del blocco 2^s

$$left^s(p) = left^{s-1}(p)$$

$$right^s(p) = \min\{right^{s-1}(p), whole^{s-1}(p+2^{s-1})\}$$

$$whole^s(p) = \min\{whole^{s-1}(p), whole^{s-1}(p+2^{s-1})\}$$

SIMD Shrinking (10)

- Mentre quando *p* è a destra

$$left^s(p) = \min\{left^{s-1}(p), whole^{s-1}(p-2^{s-1})\}$$

$$right^s(p) = right^{s-1}(p)$$

$$whole^s(p) = \min\{whole^{s-1}(p), whole^{s-1}(p-2^{s-1})\}$$

SIMD Shrinking (11)

```
procedure SIMDShrink;
        {Compute Rq for q = 2k on an SIMD hypercube}
begin
        {initialize for 20 blocks}
        whole (p) := I (p);
        left (p) := ¥;
        right (p) := ¥;
        {compute for 2i+1 blocks}
                for i :=0 to k - 1 do
                        begin
                                C (p) := whole (p);
                                C (p) ¬C (p(i));
                                left (p) := min {left (p), C (p)}; {p(i) = 1)
                                right (p) := min {right (p), C (p)}; {p(i) = 0)
                                whole (p) := min {whole (p), C (p)};
                        end
```


SIMD Shrinking (12)

```
        R (p) :=min { I (p), left (p), right (p)}
        SIMDEShift (left, -q, N);
        SIMDEShift (right, q, N);
        R (p) :=min { R(p), left (p), right (p)}
end; {of SIMDShrink}
```

- Questo algoritmo ha complessità O(log*N*) e può essere usato quando *q* non è potenza di 2 definendo

$$k = \lfloor \log q \rfloor$$

Traslazione (1)

- La traslazione consiste nello spostare il pixel dalla posizione [*i*,*j*] a [*i*+*a*,*j*+*a*] per 0≤*i*<*N* e 0≤*j*<*N* e 0≤*a* e *b*≤*N*
- La traslazione nel caso in cui *i*+*a*≥*N* e *j*+*b*≥*N* è detta ***wraparound***.
- Se invece il pixel deve essere mosso in posizione [c,d] con o *c*≥*N* o *d*≥*N*, sono scartati e i pixel della posizione [*i*,*j*] con *i*<*a* o *j*<*b* sono assegnati a 0.

Traslazione (2)

- Se *a* e *b* non sono potenze di due l'algoritmo impiega sia per un ipercubo MIMD che SIMD O(log*N*)
- Nel caso in cui siano *a* e *b* potenza di due l'algoritmo impiega O(1) in un ipercubo di tipo MIMD. Lo shift per la traslazione impiega O(| *a* |+| *b* |).

Rotazione (1)

- L'immagine I è ruotata di Θ gradi dal punto [*a*,*b*] con *a* e *b* interi e compresi tra [0, *N* - 1].
- Useremo un algoritmo di rotazione proposto da Reevers e Freancfort* in cui il pixel [*i*,*j*] sarà spostato in [*i'*,*j'*]

*A. P. Reeves and C. H. Francfort, "Data Mapping and Rotation Functions for the Massively Parallel Processor", *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, **1985**, pp. 412-417.

Rotazione (2)

$$i' = \lceil (i - a)\cos\Theta - (j - b)\sin\Theta + a \rceil$$

$$j' = \lceil (i - a)\sin\Theta - (j - b)\cos\Theta + b \rceil$$

L'equazioni per i' e j' possono essere semplificate in

$$i' = \lceil i\cos\Theta - j\sin\Theta + A \rceil$$

e

$$j' = \lceil i\sin\Theta + j\cos\Theta + B \rceil$$

Rotazione (3)

$$A = a\,(1 - \cos\Theta) + b\sin\Theta$$

e

$$B = b\,(1 - \cos\Theta) - a\sin\Theta$$

A seconda del valore di Θ, questo algoritmo è eseguito in due passi

Rotazione (4) Θ=180°

- Per Θ=180° avremo *i'*=-*i*+*a* e *j'*=-*j*+*b*

1. [Riordinamento della colonna]
   Ogni processore, *p*, imposta *dest*(*p*)=-*i*+*a* dove *i* è l'indice di riga del processore, poi l'ordinamento per colonne è fatto
2. [Riordinamento della riga]
   Ogni processore, *p*, imposta *dest*(*p*)=-*j*+*b* dove *i* è l'indice di riga del processore, poi l'ordinamento per riga è fatto

Rotazione (5) Θ=180°

- I valori di *dest* in ogni colonna al passo 1 ed in ogni riga al passo 2 sono in ordine decrescente.
- Il passo 1 manda tutti i pixel nella loro giusta posizione di destinazione per riga, il passo 2 per colonna.
- La colonna e la riordinazione di riga dei passi 1 e 2 possono essere sostituite dalla colonna e dall'inversione di riga seguite da uno Shift.

Rotazione (6) Θ=180°

- Poiché un'inversione può essere fatta nel tempo di O(log*N*), la complessità di una rotazione 180° su un ipercubo è O(logN)
- Su una mesh, la complessità è O(*N*)

Rotazione (7) Θ=±90°

- Nel caso in cui Θ=90°e Θ=-90° la soluzione è abbastanza simile
- Illustrerò solo il caso in cui Θ=90°.

- Siano :

  $i'=-j+a+b$

  e

  $j'=i-a+b$

Rotazione (8) Θ=±90°

- Avremo tre passi:

1. [Trasposta]
   Trasporre l'immagine da $I^{new}[i,j]=I^{old}[j,i]$
2. [Riordino della colonna]
   Ogni processore imposta $dest(p)=a+b-i$ quando i è il numero di riga del processore. Il riordino per colonna è fatto.
3. [Shift]
   Shift a destra di $-a+b$ è effettuato su ogni riga dell'immagine

Rotazione (9) Θ=±90°

- In una rotazione di 90° il pixel originale in posizione $[i,j]$ è ruotato di $[-j+a+b,\ i-a+b]$.
- Il passo 1 ruota il pixel nella posizione $[j,i]$;
- Il secondo passo ruota quest'ultimo $[a+b-j,i]$;
- Il passo 3 lo porta a $[a+b-j,i-a+b]$.

Rotazione (10) Θ=±90°

- La trasposizione del punto 1 può essere fatto su un ipercubo con tempo O(log*N*);
- La complessità generale su un ipercubo è O(logN).
- Il riordino della colonna al punto 2 può essere fatta da un'inversione della colonna seguita da uno shift.
- Ciò non cambia la complessità asintotica. Su un mesh, il punto 3 può essere completato nel tempo di O(N).

Rotazione (11) |Θ|≤45°

- Come in precedenza spiegherò solo $0 \le \Theta \le 45°$ poiché $-45° \le \Theta \le 0$ è simile.

1. [Riordino della colonna]
   Si imposta $dest(p) = \lceil i\cos\Theta - j\sin\Theta + A \rceil$ dove *i* è il numero di riga e *j* il numero di colonna del processore *p*. Poiché *j* è lo stesso per ogni colonna, *dest*(*p*) non è decrementato per ogni colonna.

Rotazione (12) |Θ|≤45°

Tutti i dati con la stessa direzione sono diretti a quella destinazione

Rotazione (13) |Θ|≤45°

2. [Riordino della riga]
   Imposta $dest(p) = \lceil i\tan\Theta - j\sec\Theta - A\tan\Theta + B \rceil$
   dove *i* e *j* sono rispettivamente riga e colonna del processore *p*.
3. [Shift]
   I pixel che devono essere spostati a sinistra di una posizione è spotato lungo le righe da spostare

Rotazione (14) |Θ|≤45°

- Il passo 1 invia il pixel nella sua destinazione corretta la riga.
- Da $0 \le \Theta \le 45°$ e $1/\sqrt{2} \le \cos\Theta \le 1$
- Di conseguenza, ogni processore ha al più 2 pixel diretti verso di lui.
- La colonna riordinata al passo 1, si estende fino al punto di destinazione.
- In seguito, i(il) pixel nel processo sor nella posizione [*i*,*j*] originata dal processore in colonna j e riga $\frac{i + j\sin\Theta - A - \delta}{\cos\Theta}$

Rotazione (15) |Θ|≤45°

- Quando $0 \le \delta < 1$ bisogna tenere conto della funzione ***ceiling***.
- Da questa si ottiene che i pixel sono ruotati dal processore di riga *i* e colonna $j = \lceil y \rceil$

Dove

$$y = \left(\frac{i + j\sin\Theta - A - \delta}{\cos\Theta}\right)\sin\Theta + j\cos\Theta + B =$$

$$= i\tan\Theta + j\left(\frac{\sin^2\Theta + \cos^2\Theta}{\cos\Theta}\right) - A\tan\Theta - \delta\tan\Theta + B =$$

$$= i\tan\Theta + j\sec\Theta - A\tan\Theta + B - \delta\tan\Theta$$

Rotazione (16) |Θ|≤45°

- Nel passo 2, i pixel sono prima ruotati in colonna $\lceil i\tan\Theta + j\sec\Theta - A\tan\Theta + B - \delta\tan\Theta \rceil$. Poi al passo 3, bisogna tenere conto di $\delta \tan\Theta$ della formula di y.
- Per $0 \le \Theta \le 45°$, tanΘ ha un range di[0,1].
- Quando $0 \le \delta < 1$, e $0 \le \delta\tan\Theta \le 1$, il pixel deve essere spostato a sinistra al più di una posizione.

Rotazione (16) |Θ|≤45°

- Poiché 1≤secΘ≤√2 per 0≤Θ≤45°, *dest*(*p*) è diverso per i diversi processori su alcune colonne.
- In un ipercubo la rotazione O(log*N*)
- Per un mesh O(*N*)

Rotazione (17) 0≤Θ≤360°

- Ogni Θ in un range di [0,360°] può essere visto nei seguenti modi:

a. -45°≤Θ'≤45°
b. ±90+Θ', -45°≤Θ'≤45°
c. ± 180+Θ', -45°≤Θ'≤45°

Rotazione (18) 0≤Θ≤360°

- Il caso a. lo abbiamo visto prima;
- Per la rotazione ±90° e ±180° la rotazione è stata fatta
- Quando eseguiamo una rotazione pari a Θ', ci può essere un errore soprattutto per -45°≤Θ'≤45°.
- Quest'errore può essere eliminato implementando le rotazioni di tipo ***wraparound*** ed eliminando i pixel (di wraparound) non voluti.

Scaling (1)

- Lo scaling di un'immagine di *s* per *s*≥0, intorno ad un pixel [*a*,*b*], richiede il movimento dalla posizione [*i*,*j*] a [*i'*,*j'*]

$$i' = \lceil si + a(1-s) \rceil$$
$$j' = \lceil si + b(1-s) \rceil$$
$$0 \le i$$
$$j < N$$

Scaling (2)

- Nel caso in cui *i'*≥*N* o *j'*≥*N* il pixel è scartato;
- Se due o più pixel sono mappati in un'altra posizione si hanno due casi:

1. Solo uno di questi sopravvive, la sopravvivenza di questo pixel è data da alcune operazioni come media, min, max ecc…

Scaling (3)

2. Tutti i pixel sopravvivono

- Quando s>1, spostiamo il pixel nella posizione di destinazione ed è necessario riconnettere i confini l'immagine e riempire l'oggetto in immagine.
- Lo spostamento è fatto in $O\left(\frac{(\log N)}{s}\right)$ in un ipercubo.

Scaling (4)

- Lo spostamento è fatto in $O\left(\frac{N}{s}\right)$ in un mesh quando $s<1$ e tutti i pixel di destinazione sono sopravvissuti.
- In tutti i casi lo spostamento del pixel impiega O(log*N*) su un ipecubo e O(*N*) in un mesh.
- La strategia di spostamento è eseguita prima con il riordino per riga, in seguito si riordina per colonna.
- La riconnessione dei confini dell'immagine richiede O(log*N*) su un ipecubo e O(*N*) in un mesh.

Riferimenti(1)

- [1]T. E. Chan and Y. Saad, "Multigrid algorithms on hypercube multiprocessor", *IEEE Transactions*
- *on Computers", Vol. C-35,* **Nov. 86**, pp 969-977.
- [2] E. Horowitz and S. Sahni, "*Fundamentals of Data Structures in Pascal*", Computer Science
- Press, **1985**.
- [3] S. Y. Lee, S. Yalamanchali and J. K. Agarwal, "Parallel Image Normalization on a Mesh
- **Sanjay Ranka and Sartaj Sahni**
- Connected Array Processor", *Pattern Recognition*, Vol. 20, No. 1, **Jan 87**, pp. 115-120.
- [4] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD computers", *IEEE Transactions on*
- *Computers*, No. 2, Vol. C-30, **Feb 1981**, pp. 101-107.
- [5] D. Nassimi and S. Sahni, "Optimal BPC permutations on a cube connected computer",
- *IEEE Transactions on Computers*, No. 4, Vol. C-31, **April 1982**, pp. 338-341.
- [6] V. K. Prasanna Kumar and V. Krishnan, "Efficient Image Template Matching on SIMD
- Hypercube Machines", *International Conference on Parallel Processing,* **1987**, pp 765-771.

Riferimenti(2)

- [7] S. Ranka and S. Sahni, "Image Template Matching on an SIMD hypercube multicomputer",
- *1988 International Conference on Parallel Processing*, Vol III, Algorithms & Applications,
- Pennsylvania State University Press, pp 84-91.
- [8] S. Ranka and S. Sahni, "Image Template Matching on MIMD hypercube multicomputers",
- *1988 International Conference on Parallel Processing*, Vol III, Algorithms & Applications,
- Pennsylvania State University Press, pp 92-99.
- [9] A. P. Reeves and C. H. Francfort, "Data Mapping and Rotation Functions for the Massively
- Parallel Processor", *IEEE Computer Society Workshop on Computer Architecture for Pattern*
- *Analysis and Image Database Management*, **1985**, pp. 412-417.
- [10] A. Rosenfeld and A. C. Kak, "*Digital Picture Processing*", Academic Press, **1982**
- [11] A. Rosenfeld, "A note on shrinking and expanding operations on pyramids", *Pattern Recognition*
- *Letters 6,* **Sept 1987**, pp. 241-244.
- [12] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer", *Communications*
- *of the ACM*, **1977**, pp 263-271.