Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

GENERAZIONE IN PARALLELO DI CARTE TEMATICHE DI SINTESI

Claudio MONTANI e Andrea TOMASI

Nota Interna B4-08
Marzo 1990

Abstract

The growing interest in computer graphics has to be faced with the great complexity of many practical applications. In digital mapping, for example, the operations on spatial data reach easily exponential complexity with respect to the number of regions represented on the maps, and this fact constitutes a good testbed for the research towards parallel solutions.

This paper discusses a solution in the field of parallel management of polygonal regions. The proposal is designed according to an integrated approach towards the definition of a data representation scheme suitable to be processed by parallel algorithms and to the implementation on a parallel architecture programmed by a concurrent language.

The proposed solution is based on DPCS, a distributed version of a data representation scheme called PCS. Data processing operates on partitoned data structures, and a transformation method is defined by a parallel algorithm to convert data represented as vectorial information to and from DPCS.

The DPCS system is implemented on a Transputer ring, by now composed of five Transputers, but easily extendable.

The processing and communicating facilities of the Transputers allow to exploit the maximum degree of parallelism, not only while processing the partitioned data, but also during the inherently sequential operations of data conversion.

1. Introduzione

Il crescente interesse per applicazioni grafiche sempre piu' onerose e complesse in settori quali il CAD, la cartografia automatica, la visualizzazione scientifica o l'analisi di fenomeni naturali ha evidenziato l'esigenza di disporre di strumenti efficienti per la memorizzazione, l'elaborazione e la visualizzazione di informazioni di natura geometrica.

Un esempio chiarificatore della complessita' cui possono giungere i problemi quando si trattino dati di questa natura puo' essere immediatamente derivato dalla cartografia digitale: la mappa tematica dell'uso potenziale di un suolo e' ottenibile, sotto certe condizioni, dalla sovrapposizione logica delle mappe delle acclivita', delle esposizioni dei versanti, dell'uso attuale del suolo, della insolazione, della geologia, della piovosita', etc.; in casi applicativi reali sovrapporre logicamente le carte tematiche elencate significa confrontare le centinaia di piccole regioni (rappresentanti, ognuna, un tema) di una mappa con tutte le corrispondenti regioni delle altre carte.

E' evidente che la complessita' del problema cresce esponenzialmente all'aumentare del numero di regioni distinte rappresentate in ogni mappa: una soluzione efficiente non puo' quindi prescindere dal considerare i due aspetti fondamentali del problema:

(a) lo studio di schemi di rappresentazione delle informazioni idonei al tipo di dati ed al tipo di operazioni da eseguire;

(b) la valutazione di ambienti di calcolo che consentano qualche forma di parallelismo nella grande mole di operazioni da espletare.

Pur essendo, a nostro avviso, i due aspetti del problema strettamente correlati, non molto e' stato sperimentato sino ad oggi verso una integrazione sw/hw nel trattamento di informazioni di natura spaziale.

Per quanto riguarda gli schemi di rappresentazione l'interesse della maggior parte degli autori e' stato rivolto, negli ultimi anni, verso schemi di natura gerarchica, schemi, cioe', in cui lo spazio di definizione degli oggetti da rappresentare e' suddiviso ricorsivamente in regioni regolari fino ad ottenere sottoregioni uniformi. Lo schema di rappresentazione esemplare di questa classe e' il quadtree e le sue innumerevoli derivazioni (una

bibliografia completa puo' essere trovata in [Same84] e [Same90]).

La caratteristica principale degli schemi gerarchici e' quella di sfruttare al meglio il concetto di coerenza spaziale delle informazioni raggiungendo quindi un buon compromesso tra compattezza della rappresentazione ed efficienza delle operazioni da eseguire.

Possibili forzature dell'uso indiscriminato degli schemi di rappresentazione gerarchica possono essere riscontrate quando si rappresentino informazioni logicamente troppo dissimili dalle immagini binarie per cui tali schemi sono stati definiti oppure quando si mappino le strutture dati di rappresentazione (tipicamente alberi) su modelli di computazione paralleli, generalmente array sistolici o griglie regolari di nodi di elaborazione.

La sperimentazione relativa al secondo dei punti sopracitati (elaborazione di informazioni di natura geometrica in ambienti hardware paralleli) e' relativamente giovane e, fino ad ora, orientata alla definizione di algoritmi che consentano di raggiungere un alto grado di parallelismo nelle elaborazioni logiche tra regioni poligonali, senza peraltro porsi il problema di definire opportuni schemi di rappresentazione per i dati in esame.

Esempi in questo senso possono essere ritrovati in [Kane87] (per soli poligoni a lati orizzontali o verticali) o in [Kris87]: i lavori sono accomunati dalla scelta di una architettura sistolica quale ambiente di computazione. Il lavoro [Wu89], al contrario, adotta una architettura SIMD a memoria condivisa tra i nodi di elaborazione.

In questo lavoro presentiamo un approccio *integrato* al problema, approccio che, per molti aspetti, risulta innovativo rispetto a quelli cui abbiamo poc'anzi accennato.

L'unita' logica di informazione da elaborare e' rappresentata, nel nostro caso, dall'insieme di tutte le regioni elementari del problema aventi la stessa proprieta' (o tema). In cartografia numerica questo insieme puo' rappresentare un tema di una mappa (tutte le regioni di una carta delle acclivita' con pendenza compresa tra il 10% ed il 15%), in visualizzazione tutte le aree di uno spazio planare caratterizzate da identici valori di parametri di interesse, etc.

Le regioni di una siffatta mappa binaria sono rappresentate mediante descrizione dei contorni che le limitano adottando una particolare codifica vettoriale nota come codifica in catene di Freeman [Free74].

Il modello computazionale adottato e' il classico *processor farm*: un nodo controllore impartisce i compiti e

distribuisce i dati a *p* nodi slave; ogni nodo slave invia i risultati delle operazioni richieste ad un nodo collettore che provvede alla gestione e visualizzazione dei risultati stessi.

La nostra proposta di integrazione hw/sw non si limita a distribuire i dati in forma vettoriale tra i diversi nodi di elaborazione adottando semplicemente qualche tecnica di bilanciamento di carico: in modo parallelo i contorni delle regioni in ingresso sono convertiti in un nuovo schema di rappresentazione denominato *Distributed Parallel Connected Stripes (DPCS)* che consente di parallelizzare le successive operazioni logiche tra mappe distinte (e non gia' tra due distinte regioni) con un grado di parallelismo direttamente proporzionale al numero di processori slave presenti nell'architettura.

Sempre in maniera altamente parallela i risultati delle operazioni sono restituiti, con schema di rappresentazione vettoriale, al nodo collettore del sistema.

Lo schema di rappresentazione *DPCS* (estensione in ambito parallelo del corrispondente schema *PCS* [Mont84]) si presta immediatamente ad una suddivisione delle mappe da elaborare in larghe striscie orizzontali: ogni striscia sara' gestita ed elaborata da un singolo nodo slave del sistema. Questa politica di suddivisione delle informazioni, peraltro gia' sperimentata nell'ambito della elaborazioni delle immagini ([Arab87] e [Arab89]) si dimostra ottimale in quanto presenta una logica corrispondenza tra l'architettura del sistema e relative comunicazioni tra i nodi e la correlazione spaziale propria delle regioni rappresentate in una mappa.

Lo schema *DPCS* rende le operazioni scalari e logiche su mappe binarie veramente semplici e dirette; le mappe tematiche come descritte nell'esempio iniziale vengono rappresentate nel sistema come l'insieme di *n* mappe binarie rappresentanti, ognuna, un tema della carta originale.

Il capitolo 2 presenta le caratteristiche principali dello schema di codifica *PCS*, fornisce una dettagliata descrizione degli algoritmi per la conversione da e verso la rappresentazione vettorialee ed accenna ai semplici algoritmi per effettuare operazioni logiche o scalari sui dati.

Il capitolo successivo propone il nostro modello di computazione distribuito e definisce lo schema *DPCS* per la rappresentazione ed elaborazione di informazioni spaziali sull'architettura proposta. Viene anche discussa e descritta la versione parallela degli algoritmi operanti sulla struttura dati e gia' considerati in ambito sequenziale.

Il capitolo 4 illustra dettagliatamente la

realizzazione del sistema proposto su una architettura ad anello di Transputer INMOS [INMO87a], [INMO87b], [INMO88a] e [INMO88b]. Si sottolinea come possa essere emulata l'architettura processor farm teorica definita al capitolo 3 con una corrispondenza diretta sulla architettura di Transputer.

I capitoli 5 e 6, infine, forniscono indicazioni sulle prestazioni del sistema e sulle possibilita di ampliamenti e miglioramenti successivi.

2. La struttura dati PCS

2.1 Generalita'

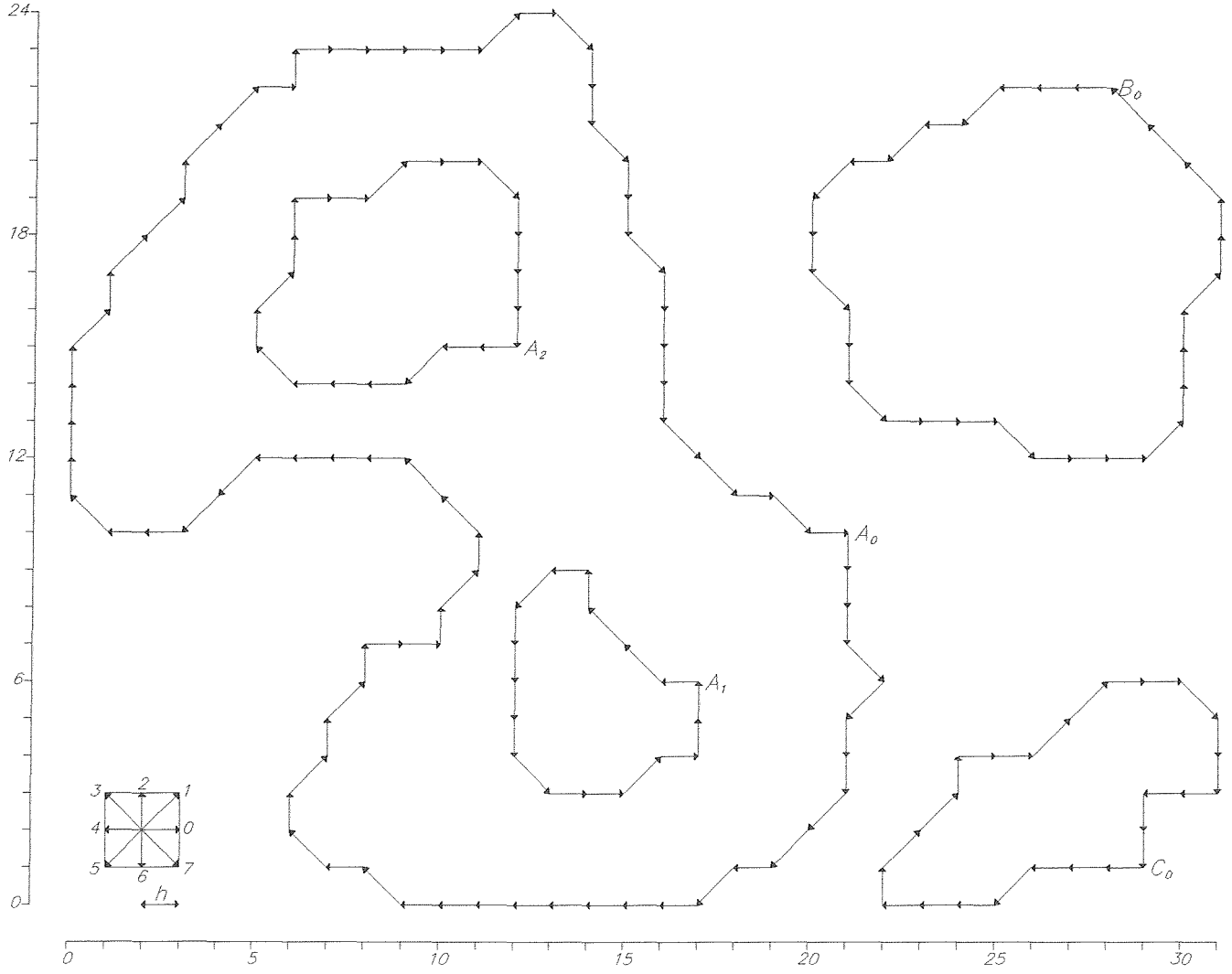
Nel 1984 uno degli autori defini' la struttura dati PCS (Parallel Connected Stripes) quale nuovo schema di codifica per la rappresentazione efficiente di superfici poligonali [Mont84]. La struttura dati consente la gestione di mappe binarie (ad esempio un tema di una carta tematica) e si rivela particolarmente funzionale nelle elaborazioni logiche (unioni, intersezioni, etc.) tra mappe distinte.

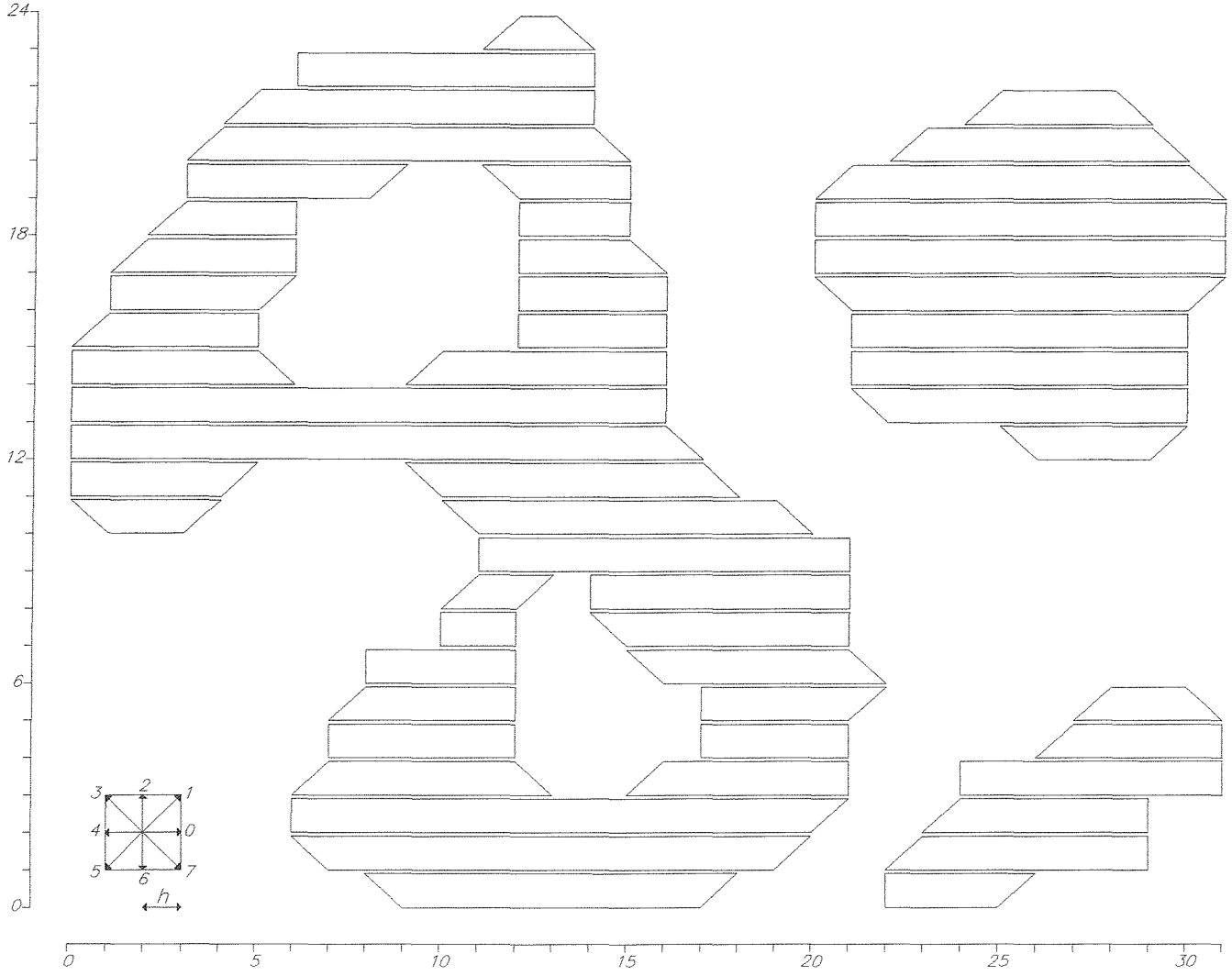
Per sua natura lo schema PCS appartiene alla classe degli schemi di rappresentazione di tipo raster [Merr73] ma presenta la peculiarita' di sostituire al concetto di linea di scansione del metodo raster classico il concetto di striscia di scansione.

2.2 La struttura dati PCS

Illustriamo con un esempio le principali caratteristiche dello schema PCS. In Fig. 2.1 e' rappresentata una mappa binaria costituita da tre regioni poligonali. Ogni contorno di regione (o frontiera di buco) e' rappresentato mediante una catena di Freeman, cioe' tramite le coordinate cartesiane del punto di inizio della catena ed una sequenza di link descriventi spostamenti elementari lungo i nodi di un grigliato regolare idealmente sovrapposto alla mappa. Come mostrato in figura i possibili spostamenti sono 8 e sono relativi al passo h prefissato.

Per motivi legati alla compattezza di memorizzazione ed alla rappresentabilita' su periferiche grafiche di tipo vettoriale (ad esempio plotter), i dati di ingresso al nostro problema ed i risultati delle elaborazioni saranno codificati nello schema vettoriale a catene appena descritto. L'eventuale conversione da regioni poligonali classiche (cosi' come restituite da digitalizzatore) a catene di Freeman e' una semplice operazione descritta


24
18
12
6
0
0
5
10
15
20
25
30
A_0
A_1
A_2
B_0
C_0
3
2
1
4
0
5
6
7
h


24
18
12
6
0
0
5
10
15
20
25
30
1
2
3
4
5
6
7
0
h

esaurientemente in [Free74].

Supponiamo di tagliare la mappa di Fig. 2.1, lungo le linee orizzontali del grigliato, in striscie di spessore h e di ricavare da queste le sottostriscie appartenenti alle regioni in esame. L'insieme delle sottostriscie ottenute con questo procedimento e' mostrato in Fig. 2.2 e rappresenta la codifica PCS delle regioni di Fig. 2.1. Una completa descrizione di ogni striscia di Fig. 2.2 puo' essere ottenuta indicando semplicemente l'ordinata y della striscia (abbiamo scelto di indicare l'ordinata del lato inferiore della striscia) ed il numero di sottostriscie che le appartengono. Ciascuna sottostriscia, di forma trapezoidale o triangolare (nei punti di massimo o minimo locali) puo' essere descritta dal tipo (una di tre possibili inclinazioni) e dal valore dell'ascissa dei suoi lati ovest ed est. In Fig. 2.3 sono mostrati i possibili tipi di lato di sottostriscia ed il corrispondente valore dell'ascissa con riferimento al passo del grigliato adottato.

L'informazione rappresentata graficamente in Fig. 2.2 e' descritta in forma digitale in Tab. 2.1. La tabella evidenzia, per ogni sottostriscia, i valori delle ascisse e, con numeri indice, il tipo di lato.

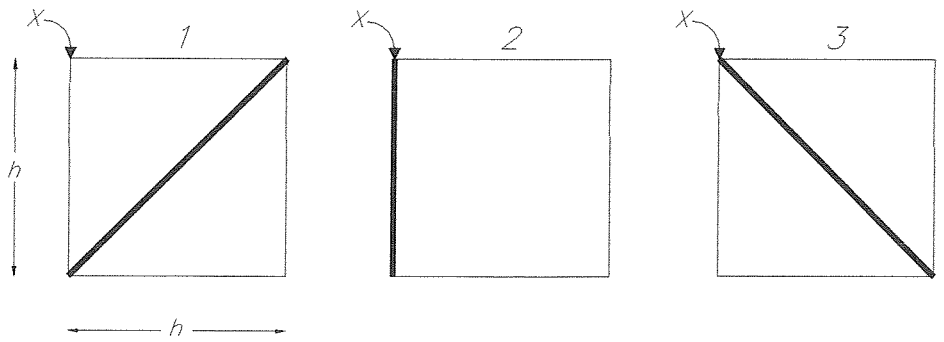
2.3 Conversione da Catene di Freeman a PCS

Le Figg. 2.1 e 2.2 mostrano chiaramente che ogni lato (ovest o est) di una generica sottostriscia corrisponde ad un link non orizzontale della catena di Freeman in ingresso. Il processo di conversione da catene a PCS e' appunto legato a questa osservazione. Ogni catena di Freeman viene scandita sequenzialmente: per ogni link non orizzontale un lato di sottostriscia e' inserito nella corrispondente striscia rispettando un ordine crescente per ascissa. Alla fine del procedimento, i lati ovest ed est di ogni sottostriscia saranno individuati dalle coppie di lati in posizione dispari e pari rispettivamente.

Lo pseudo codice di Fig. 2.4 illustra l'algoritmo di conversione (*Chain_to_PCS*) da catena di Freeman a PCS. La procedura converte la catena di Freeman appartenente alla mappa M:

$$M, x_0, y_0, nl, L[0 \div nl-1]$$

(in cui x_0, y_0 rappresenta il punto di partenza in coordinate del grigliato, nl il numero di link della catena e $L[0 \div nl-1]$ e' il vettore di link) in lati di sottostriscie da inserire nella struttura dati PCS. Per ogni link non orizzontale la procedura *Side_Insert(M,x,y,t)* inserisce un lato di sottostriscia di tipo t nella striscia di ordinata y della mappa M rispettando un ordinamento crescente in x.


X
1
h
h
X
2
X
3

Stripe Ord	Number of Substripes	Substripe[1] West	Substripe[1] East	Substripe[2] West	Substripe[2] East	Substripe[3] West	Substripe[3] East
23	1	11^{1}	13^{3}				
22	1	6^{2}	14^{2}				
21	2	4^{1}	14^{2}	24^{1}	28^{3}		
20	2	3^{1}	14^{3}	22^{1}	29^{3}		
19	3	3^{2}	8^{1}	11^{3}	15^{2}	20^{1}	29^{3}
18	3	2^{1}	6^{2}	12^{2}	15^{2}	20^{2}	31^{2}
17	3	1^{1}	6^{2}	12^{2}	15^{3}	20^{2}	31^{1}
16	3	1^{2}	5^{1}	12^{2}	16^{2}	20^{3}	30^{1}
15	3	0^{1}	5^{2}	12^{2}	16^{2}	21^{2}	30^{2}
14	3	0^{2}	5^{3}	9^{1}	16^{2}	21^{2}	30^{2}
13	2	0^{2}	16^{2}	21^{3}	30^{2}		
12	2	0^{2}	16^{3}	25^{3}	29^{1}		
11	2	0^{2}	4^{1}	9^{3}	17^{3}		
10	2	0^{3}	3^{1}	10^{3}	19^{3}		
9	1	11^{2}	21^{2}				
8	2	10^{1}	12^{1}	14^{2}	21^{2}		
7	2	10^{2}	12^{2}	14^{3}	21^{2}		
6	2	8^{2}	12^{2}	15^{3}	20^{3}		
5	3	7^{1}	12^{2}	17^{2}	21^{1}	27^{1}	30^{3}
4	3	7^{2}	12^{2}	17^{2}	21^{2}	26^{1}	31^{2}
3	3	6^{1}	12^{3}	15^{1}	21^{2}	24^{2}	31^{2}
2	2	6^{2}	20^{1}	23^{1}	29^{2}		
1	2	6^{3}	19^{1}	22^{1}	29^{2}		
0	2	8^{3}	17^{1}	22^{2}	25^{1}		

Table 2.1

```
procedure Chain_to_PCS()
    x ← x0
    y ← y0
    for k = 0, k = k+1 while k < nl do
        switch(L[k])
            case 0:
                x ← x+1
            case 1:
                Side_Insert(M,x,y,1)
                x ← x+1
                y ← y+1
            case 2:
                Side_Insert(M,x,y,2)
                y ← y+1
            case 3:
                x ← x-1
                Side_Insert(M,x,y,3)
                y ← y+1
            case 4:
                x = x-1
            case 5:
                x = x-1
                y = y-1
                Side_Insert(M,x,y,1)
            case 6:
                y = y-1
                Side_Insert(M,x,y,2)
            case 7:
                y = y-1
                Side_Insert(M,x,y,3)
                x = x+1
end procedure
```

Fig. 2.4

L'eventuale inserimento di due lati identici nella struttura (identici ascissa, ordinata e tipo di lato) porta alla cancellazione di entrambi dalla struttura stessa.

2.4 Conversione da PCS a Catene di Freeman

Il processo inverso, la conversione da PCS a catene di Freeman e' altrettanto semplice: e' possibile *camminare* da una sottostriscia di ordinata *y* ad una di ordinata *y+h* o *y-h* soltanto se queste risultano connesse da una base (o parte di essa) in comune; e' possibile *camminare* da una sottostriscia di ordinata y ad un'altra di identica ordinata soltanto se esiste una sottostriscia di ordinata *y+h* o *y-h* che le connette. Le procedure delle figure 2.5 (*PCS_to_Chain*), 2.6 (*Bottom_Up*) e 2.7 (*Top_Down*) illustrano le diverse fasi del processo di ricostruzione. Per semplicita' abbiamo omesso l'indicazione della mappa *M* cui gli algoritmi si riferiscono.

La routine *PCS_to_Chain* ricerca iterativamente un lato di sottostriscia non gia' interessato dal processo di conversione (un lato cioe' non ancora marcato); se il lato e' un lato ovest di sottostriscia la procedura attiva *Bottom_Up* sulla sottostriscia in questione per la ricostruzione in senso orario del contorno di una regione. Se, al contrario, il lato ovest di una sottostriscia risulta marcato ed il lato est non ancora, *PCS_to_Chain* attiva *Top_Down* sulla sottostriscia per la ricostruzione in senso antiorario del contorno di un buco. Il procedimento ha termine quando la struttura relativa alla mappa da convertire (di extents *YMIN e YMAX*) e' stata completamente visitata.

Le procedure di ricostruzione fanno uso di routine elementari che operano sulla struttura PCS: *Ss_Num(y)* restituisce il numero di sottostriscie appartenti alla striscia di ordinata *y*; $Absc(S_r^y,p)$ restituisce l'ascissa del vertice *p* (*p* assume valore *nw*, *ne*, *sw* oppure *se*) della *r*-esima sottostriscia della striscia di ordinata *y*; $Type(S_r^y,d)$ restituisce il tipo di lato del lato *d* (*d* assume valori *w* oppure *e*) della *r*-esima sottostriscia della striscia di ordinata *y*; $Check\_mark(S_r^y,d)$ restituisce *True* se il lato *d* (*d* assume valori *w* oppure *e*) della *r*-esima sottostriscia della striscia di ordinata *y* e' gia' stato visitato durante il processo di conversione da PCS a Catene di Freeman, *False* altrimenti; $Mark\ (S_r^y,d)$ imposta al valore *True* la condizione di lato *d* (*d* assume valori *w* oppure *e*) della *r*-esima sottostriscia della striscia di ordinata *y* marcato.

La procedura *Display_Chain* e' dipendente dalla implementazione (ogni catena puo' essere visualizzata o

```
procedure PCS_to_Chain( )
    for y ← YMIN, y ← y+1 while y < YMAX do
        for r ← 0, r ← r+1 while r < Ss_Num(y) do
            if (Check_Mark(S_r^y,w) = False) then
                x_0 ← Absc(S_r^y,sw)
                y_0 ← y
                nl = 0
                Bottom_Up(S_r^y)
                Display_Chain(x_0,y_0,nl,L[0 ÷ nl-1])
            else if (Check_Mark(S_r^y,e) = False) then
                x_0 ← Absc(S_r^y,ne)
                y_0 ← y+1
                nl = 0
                Top_Down(S_r^y)
                Display_Chain(x_0,y_0,nl,L[0 ÷ nl-1])
end procedure
```

Fig. 2.5

```
procedure Bottom_Up (S_r^y)
    if (Check_Mark(S_r^y,w) = True) then
        return
    else
(a)     L[nl] ← Type(S_r^y,w)
        nl ← nl+1
        Mark(S_r^y,w)
        for k ← 0, k ← k+1 while k < Ss_Num(y+1) do
            if ((Absc(S_r^y,nw) < Absc(S_k^{y+1},se)) and
                                (Absc(A_r^y,ne) > Absc(S_k^{y+1},sw))) then
                X_s ← Absc(S_k^{y+1},sw)
                if (Absc(S_r^y,nw) ≤ X_s) then
(c)                 for l ← Absc(S_r^y,nw), l ← l+1 while l < X_s do
                        L[nl] ← 0
                        nl ← nl+1
(b)                 Bottom_Up(S_k^{y+1})
                    return
                else if ((r = 0) or (Absc(S_{r-1}^y,ne) ≤ X_s)) then
(g)                 for l ← X_s, l ← l+1 while l < Absc(S_r^y,nw) do
                        L[nl] ← 4
                        nl ← nl+1
                    Bottom_Up(S_k^{y+1})
                    return
                else
(f)                 for l ← Absc(S_{r-1}^y,ne), l ← l+1 while l < Absc(S_r^y,nw) do
                        L[nl] ← 4
                        nl ← nl+1
                    Top_Down(S_{r-1}^y)
                    return
(e)     for l ← Absc(S_r^y,nw), l ← l+1 while l < Absc(S_r^y,ne) do
            L[nl] ← 0
            nl ← nl+1
        Top_Down(S_r^y)
        return
end procedure
```

Fig. 2.6

```
procedure Top_Down(S_r^y)
    if (Check_Mark(S_r^y, e) = True) then
        return
    else
(a)     L[nl] ← Type(S_r^y, e) + 4
        nl ← nl+1
        Mark(S_r^y, e)
        for k ← Ss_Num(y-1)-1, k ← k-1 while k ≥ 0 do
            if ((Absc(S_r^y, sw) < Absc(S_k^{y-1}, ne)) and
                                (Absc(A_r^y, se) > Absc(S_k^{y-1}, nw))) then
                X_s ← Absc(S_k^{y-1}, ne)
                if (Absc(S_r^y, se) ≥ X_s) then
(c)                 for l ← X_s, l ← l+1 while l < Absc(S_r^y, se) do
                        L[nl] ← 4
                        nl ← nl+1
(b)                 Top_Down(S_k^{y-1})
                    return
                else if ((r = Ss_Num(y)-1) or (Absc(S_{r+1}^y, sw) ≥ X_s)) then
(g)                 for l ← Absc(S_r^y, se), l ← l+1 while l < X_s do
                        L[nl] ← 0
                        nl ← nl+1
                    Top_Down(S_k^{y-1})
                    return
                else
(f)                 for l ← Absc(S_r^y, se), l ← l+1 while l < Absc(S_{r+1}^y, se) do
                        L[nl] ← 0
                        nl ← nl+1
                    Bottom_Up(S_{r+1}^y)
                    return
(e)     for l ← Absc(S_r^y, sw), l ← l+1 while l < Absc(S_r^y, se) do
            L[nl] ← 4
            nl ← nl+1
        Bottom_Up(S_r^y)
        return
end procedure
```

Fig. 2.7

memorizzata).

La procedura ricorsiva *Bottom_Up* (Fig. 2.8), dopo aver prodotto il link relativo al lato ovest della sottostriscia (r di ordinata y) in esame, analizza la striscia di ordinata immediatamente superiore (con scansione da sinistra a destra) alla ricerca di una sottostriscia superiormente connessa. Se tale sottostriscia (k) esiste (Fig. 2.8b e 2.8c) ed e' in posizione tale che:

$$Absc(S_r^y, nw) \leq Absc(^{y+1}_k, sw)$$

si procede in modo bottom_up sulla sottostriscia k dopo aver eventualmente generato link 0 di raccordo. Se esiste una sottostriscia superiormente connessa ma vale la relazione (Fig. 2.8d):

$$Absc(S_r^y, nw) > Absc(^{y+1}_k, sw)$$

la procedura analizza la posizione della eventuale sottostriscia $r-1$ di ordinata y. Nel caso di Fig. 2.8f vengono generati eventuali link 4 di raccordo e si attiva la procedura *Top_Down* sulla sottostriscia S_{r-1}^y; nel caso di Fig. 2.8g, invece, la ricerca continua in modo bottom_up su S_k^{y+1}. Se, infine, non esistono sottostriscie superiormente connesse alla sottostriscia in esame (Fig. 2.8e) il verso di movimento e' invertito e si attiva *Top_Down* sulla sottostriscia stessa.

Discorso speculare ma in tutto identico puo' essere fatto per la procedura *Top_Down* (Fig. 2.9). La Fig. 2.10 mostra le catene di Freeman prodotte dall'applicazione dell'algoritmo alla mappa di Fig. 2.2. Essa evidenzia anche la sequenza di routine invocate ed i link da queste prodotti.

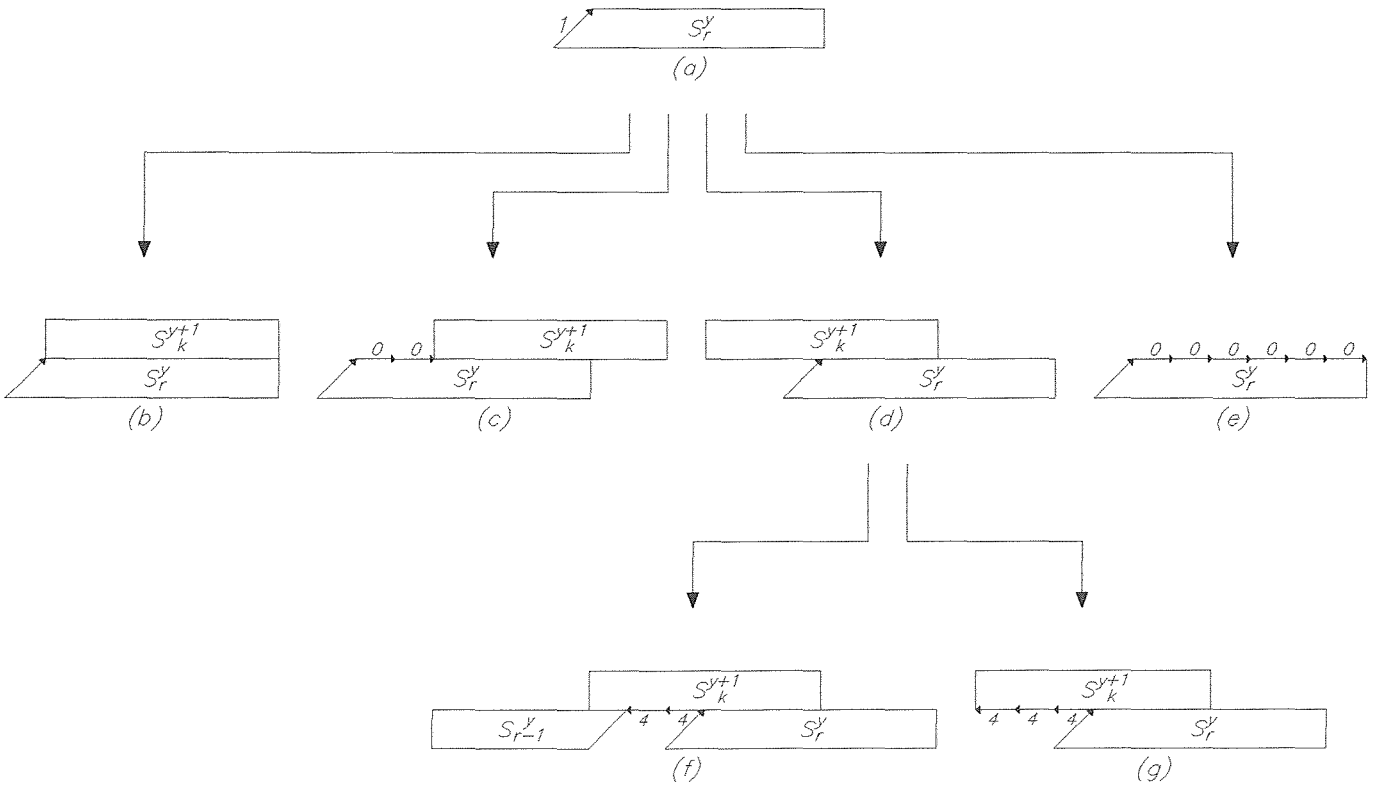
Deve essere sottolineato come il processo di conversione da PCS a catene di Freeman restituisca il contorno delle regioni in senso orario, il contorno dei buchi in senso antiorario; questa caratteristica e' molto importante ai fini di una corretta visualizzazione dei risultati.

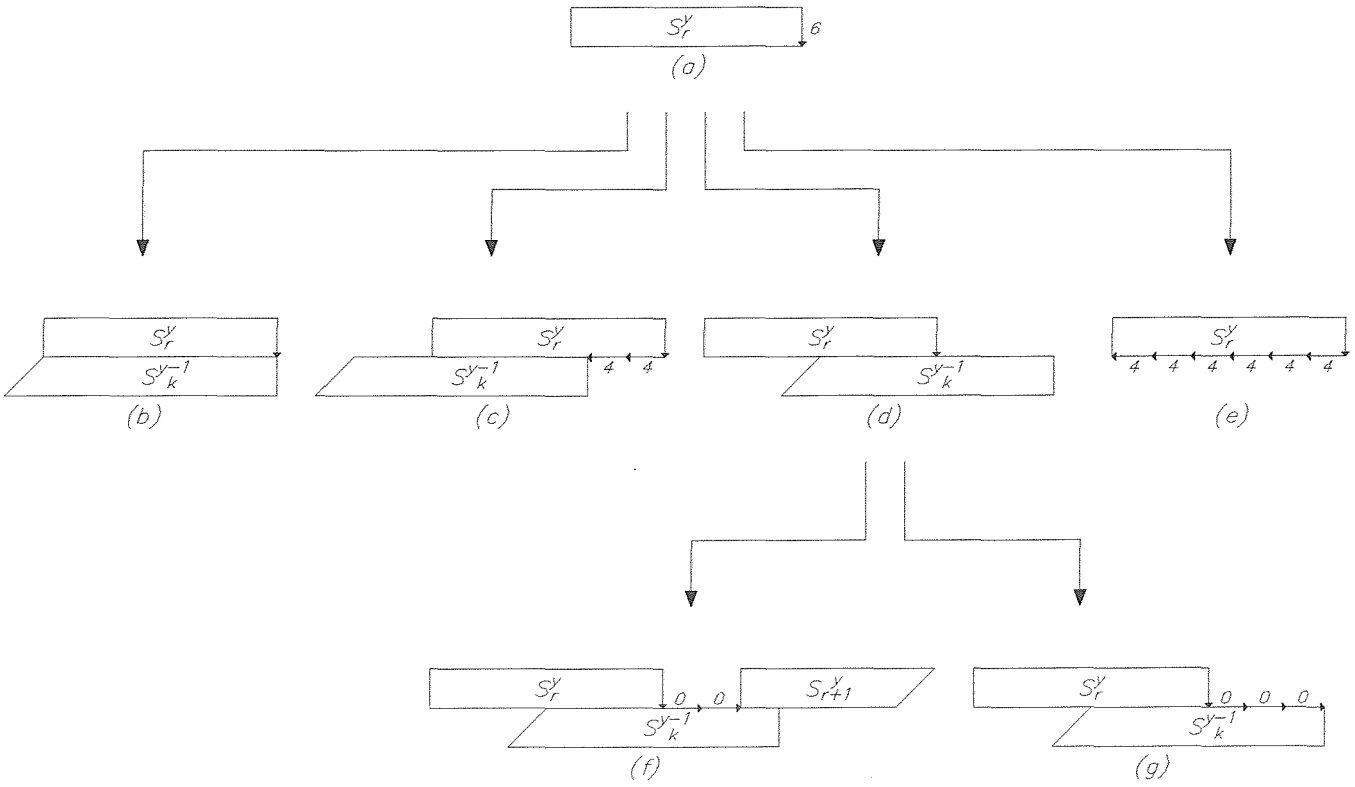
2.5 Le operazioni scalari e logiche

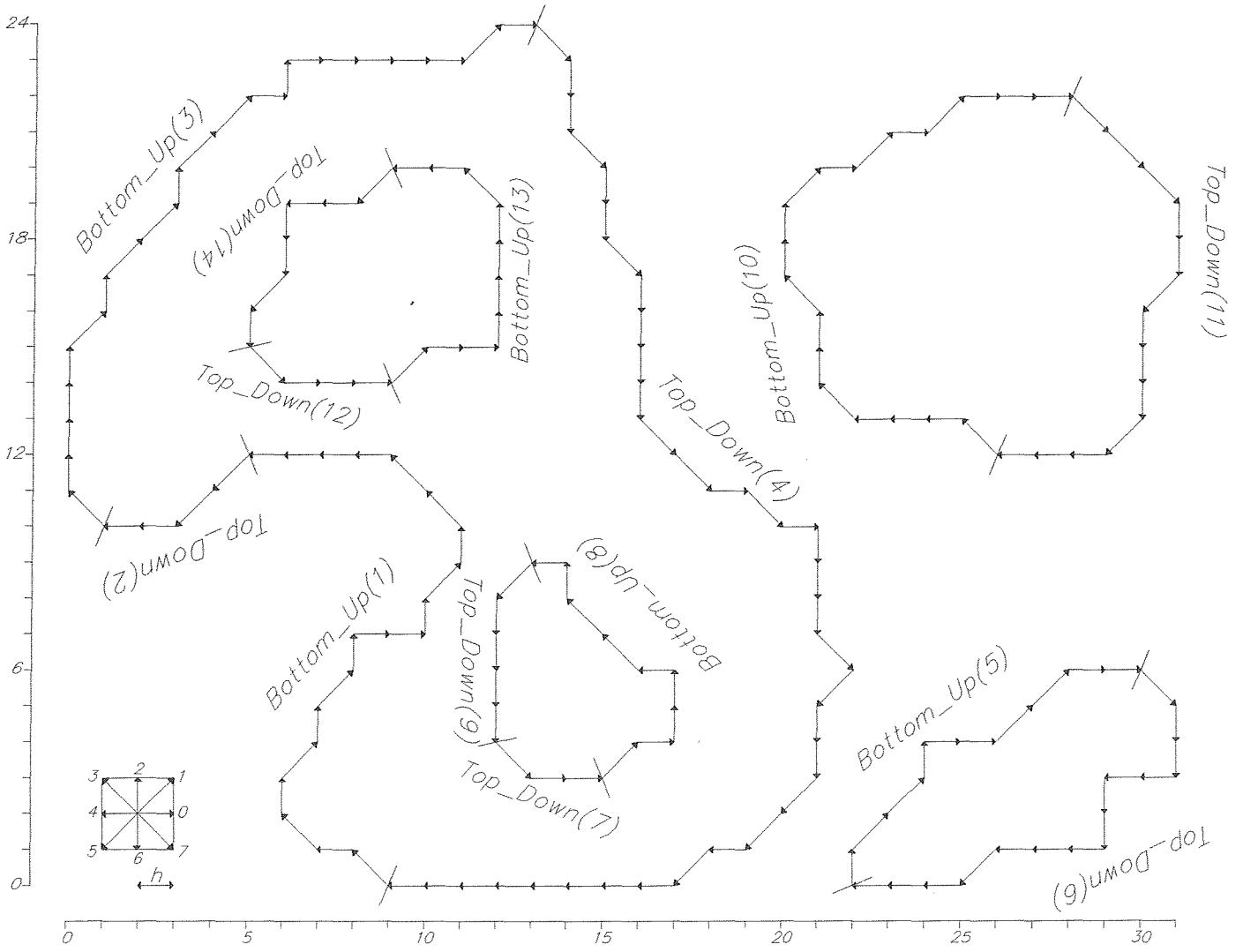
Possiamo classificare le operazioni effettuabili su mappe in operazioni di tipo scalare (appartenenza di un punto ad una regione, area di una mappa, perimetro, etc.) applicabili ad una singola mappa ed operazioni di tipo logico (unione, intersezione, etc.) coinvolgenti due o piu' mappe.

Gli algoritmi per la realizzazione delle operazioni di tipo scalare sono immediatamente derivabili. In particolare:

(a) l'appartenenza di un punto $P \equiv (x,y)$ ad una regione e'


1
S_r^y
(a)
S_k^{y+1}
S_r^y
(b)
0 0
S_k^{y+1}
S_r^y
(c)
S_k^{y+1}
S_r^y
(d)
0 0 0 0 0 0
S_r^y
(e)
S_k^{y+1}
S_{r-1}^y
4 4
S_r^y
(f)
S_k^{y+1}
4 4 4
S_r^y
(g)


S_r^y
6
(a)
S_r^y
S_k^{y-1}
(b)
S_r^y
S_k^{y-1}
4
4
(c)
S_r^y
S_k^{y-1}
(d)
S_r^y
4
4
4
4
4
4
(e)
S_r^y
S_k^{y-1}
0
0
S_{r+1}^y
(f)
S_r^y
S_k^{y-1}
0
0
0
(g)


Bottom_Up(3)
Top_Down(14)
Bottom_Up(13)
Top_Down(12)
Top_Down(2)
Bottom_Up(1)
Top_Down(9)
Bottom_Up(8)
Top_Down(7)
Top_Down(4)
Bottom_Up(10)
Top_Down(11)
Bottom_Up(5)
Top_Down(6)
3
2
1
4
0
5
6
7
h
0
5
10
15
20
25
30
0
6
12
18
24

verificata controllando l'esistenza di una sottostriscia di ordinata $\lceil y/h \rceil$ che contenga l'ascissa x/h;

(b) l'area complessiva di una mappa e' ottenibile visitando iterativamente la struttura dati PCS e sommando, per ogni sottostriscia S, il contributo:

$$(Absc(S,e)-Absc(S,w) * h + \\ - h^2/2 * |Type(S,w) - 2| + \\ + h^2/2 * |Type(S,e) - 2|$$

Nella formula il secondo e terzo termine sono relativi ai contributi di $\pm h^2/2$ dovuti ai diversi tipi di lato ovest ed est della sottostriscia.

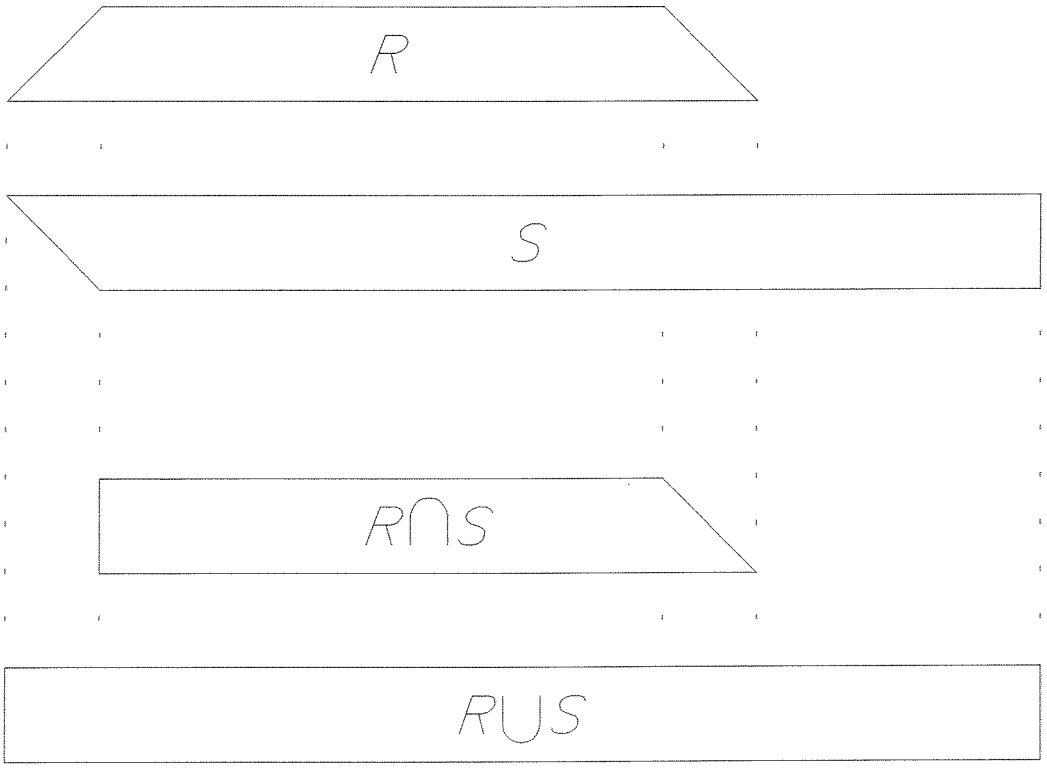
(c) il perimetro dei contorni delle regioni di una mappa e' ottenuto "percorrendo" i contorni stessi. Una applicazione dell'algoritmo di conversione da struttura dati PCS a catene di Freeman che non produca link in uscita ma semplicemente conti gli spostamenti effettuati consente di calcolare il perimetro complessivo di una mappa.

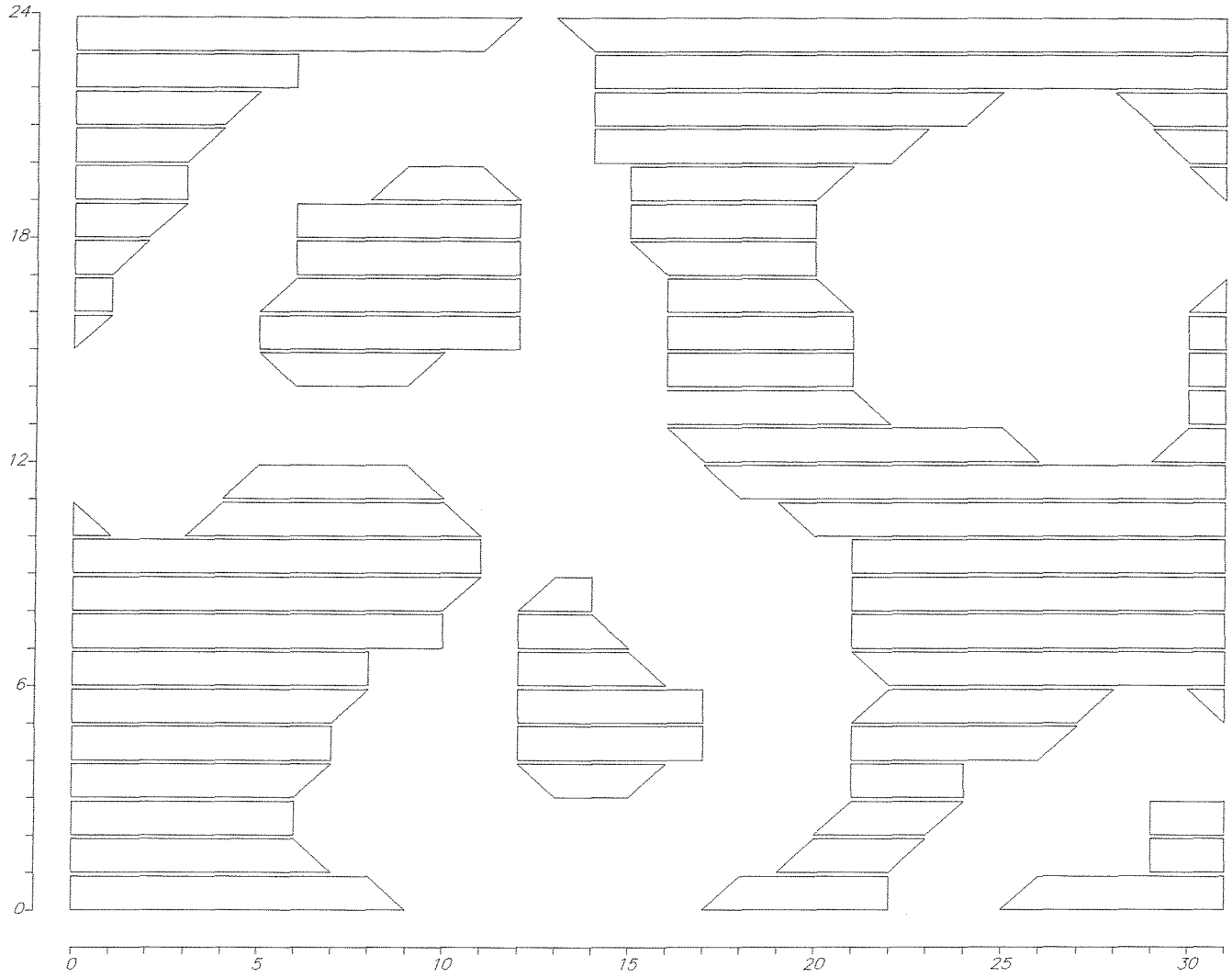
La struttura dati PCS e' particolarmente adatta alla rappresentazione di mappe topologicamente molto complesse (regioni concave o convesse con o senza nidificazioni di buchi). Dal momento che il PCS e' uno schema di rappresentazione raster, le operazioni logiche tra mappe distinte consistono nel confrontare iterativamente le striscie di uguale ordinata (scansione parallela) e, per ogni coppia di striscie, le corrispondenti sottostriscie. Il confronto tra due sottostriscie si riduce ad un confronto tra intervalli lineari anche se una particolare attenzione deve essere prestata nella determinazione dei lati della sottostriscia risultato di una operazione. Come illustrato in Fig. 2.11 nei casi di ambiguita' (unione o intersezione delle sottostriscie S e R di figura) abbiamo adottato la soluzione che mantenesse la congruenza dei tipi di lato di sottostriscia.

Un cenno particolare merita l'algoritmo per il complemento di una mappa. La negazione di una mappa puo' essere ricondotta alla differenza tra lo spazio di definizione delle mappe (il rettangolo *[XMIN, XMAX, YMIN, YMAX]* rappresentante gli extents delle mappe) e la mappa stessa. Da un punto di vista algoritmico il complemento e' ottenuto inserendo nella mappa in esame due lati di sottostriscia di tipo 2 ed ascisse *XMIN* e *XMAX* rispettivamente per tutte le striscie di ordinata compresa tra *YMIN* e *YMAX*.

Come mostra la Fig. 2.12 (mappa complemento della mappa di Fig. 2.2 rispetto al rettangolo *[0,31,0,24]*) l'inserzione dei nuovi lati fa si' che i lati di sottostriscia di posto pari diventino di posizione dispari e viceversa, il "dentro" delle regioni diventi il "fuori" e viceversa.

L'adozione del concetto di striscia di scansione


R
S
R∩S
R∪S


24
18
12
6
0
0
5
10
15
20
25
30

anziche' quello di linea di scansione assicura che i risultati delle operazioni logiche ed i processi di conversione da rappresentazione vettoriale a PCS e viceversa non presentino ambiguita'.

3. La soluzione distribuita

3.1 Generalita'

Abbiamo mostrato nella sezione precedente come la natura raster dello schema di codifica PCS consenta di compiere operazioni logiche tra mappe distinte considerando semplicemente striscie di uguale ordinata. Le operazioni su due striscie di identica ordinata non necessitano di alcuna informazione che non sia direttamente ottenibile dalle striscie stesse. Ne consegue immediatamente che la disponibilita' di un ambiente hardware multiprocessore in cui ogni nodo operasse su un certo numero di striscie porterebbe ad un incremento della performance del sistema direttamente proporzionale al numero di processori.

Da un punto di vista teorico le considerazioni precedenti non possono essere estese al processo completo di generazione di una mappa in quanto gli algoritmi di conversione da catene di Freeman a PCS e viceversa sono prettamente sequenziali; le catene stesse rappresentano uno schema di codifica sequenziale: la posizione sul piano dello spostamento prodotto da ogni link e' dipendente dal valore dei link precedenti.

Operando in ambiente distribuito, si puo' pensare di mantenere ancora gli stessi algoritmi sequenziali, la cui elaborazione avviene nell'interfaccia tra il sistema distribuito e l'esterno, facendo di conseguenza circolare nel sistema solo dati gia' rappresentati in codifica PCS. In tal caso l'operazione di conversione non e' passibile di ottimizzazioni: una sola catena alla volta viene convertita, in modo rigorosamente sequenziale. In alternativa, e' possibile adottare un diverso approccio per la conversione, facendo uso di algoritmi paralleli per sfruttare al meglio le caratteristiche dell'architettura distribuita.

E' opportuno sottolineare che ogni mappa reale contiene decine di regioni per ogni tema che la mappa rappresenta: nell'ottica di un ambiente multiprocessore come quello sopra accennato, ogni nodo potrebbe gestire ed elaborare le catene (verso e da PCS) topologicamente piu' prossime alle striscie in carico al nodo stesso.

Il parallelismo che si riesce ad esplicitare consiste nell'elaborazione in parallelo di catene diverse e

nell'elaborazione in pipeline di porzioni della stessa catena su nodi diversi del sistema distribuito.

3.2 La soluzione distribuita

E' sulla base delle precedenti considerazioni che abbiamo ipotizzato una soluzione distribuita di un sistema per la elaborazione di mappe tematiche di sintesi basata su una architettura MIMD come schematizzata in Fig. 3.1. Ogni nodo della rete e' un processore general purpose dotato di propria memoria programmi/dati ed in grado di comunicare con gli altri nodi grazie alla rete di interconnessione indicata in figura. Le comunicazioni avvengono mediante scambio di messaggi; ogni messaggio, come vedremo, e' costituito da richieste di elaborazione (comandi) e da informazioni.

Nel sistema sono presenti due nodi dedicati, con il compito di interfacciare l'architettura parallela verso il mondo esterno. Il nodo *C* costituisce l'interfaccia in ingresso, per l'immissione delle mappe da elaborare e di comandi per l'attivazione delle funzioni di cui il sistema dispone; il nodo *D* e' l'interfaccia in uscita, per la visualizzazione e memorizzazione dei risultati.

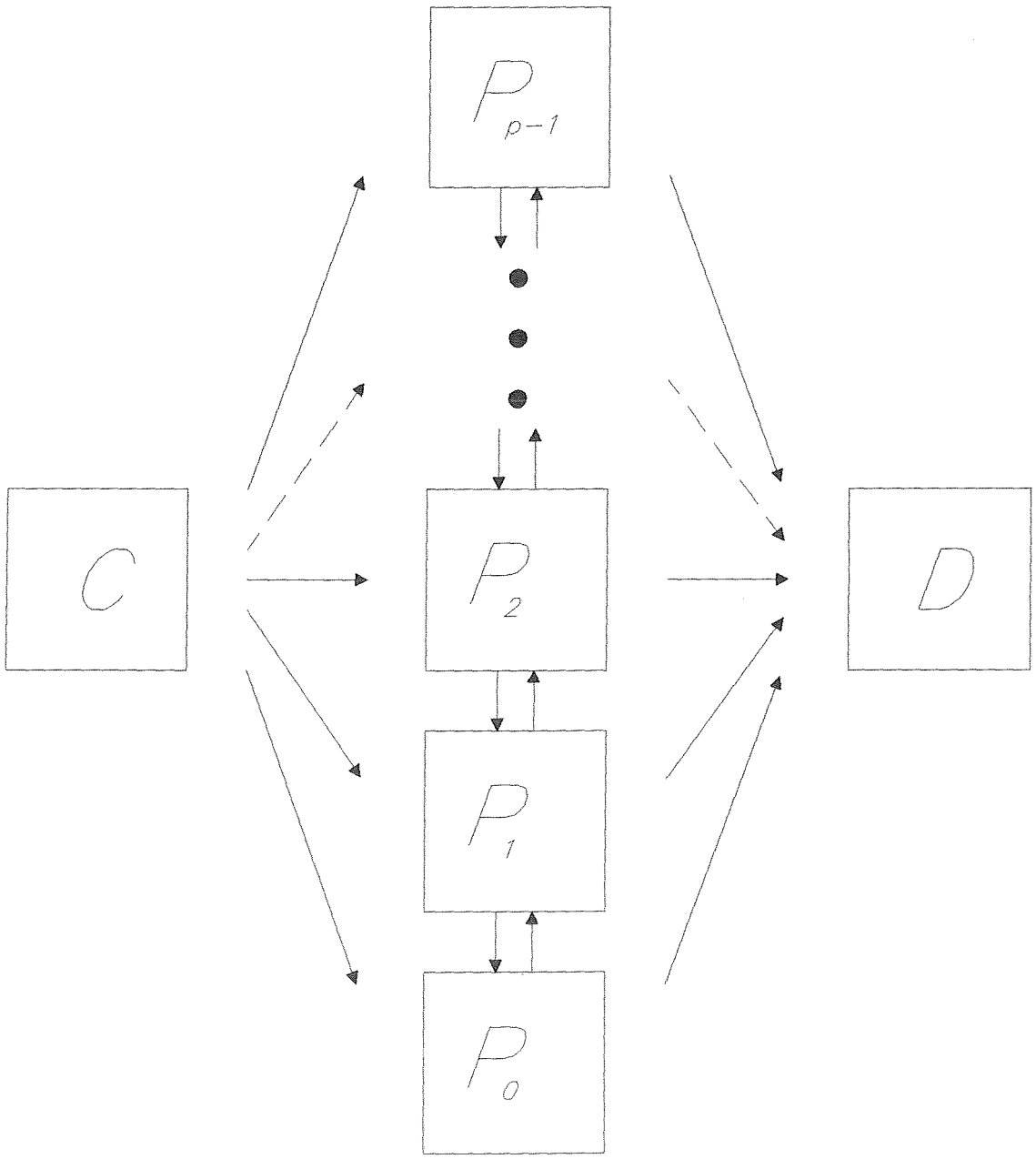
La strategia adottata e' molto semplice: sulla base degli extents delle mappe da elaborare [*YMIN,YMAX*] il nodo *C* assegna a ciascuno dei *p* nodi $P_0, P_1, \ldots, P_{p-1}$ un intervallo di ordinate $[YMIN_i, YMAX_i]$ $i=0,1,\ldots,p-1$. Ciascun intervallo rappresenta lo spazio in *Y* delle striscie gestite ed elaborate da ciascun nodo P_i.

Nella presunzione di regioni uniformemente distribuite abbiamo assunto, in prima ipotesi, che tutti gli intervalli presentino la medesima ampiezza (*1/p* dell'ampiezza in *Y* delle mappe da elaborare). Con riferimento al grigliato regolare idealmente sovrapposto alle regioni, valgono le relazioni:

$$YMIN_i = YMAX_{i-1} + 1$$
$$YMAX_i = YMIN_{i+1} - 1$$

Le macro-operazioni che il sistema deve eseguire si possono riassumere in: (1) conversione dalle catene di Freeman allo schema di codifica PCS distribuito, (2) elaborazioni logiche tra mappe o scalari su una mappa e (3) restituzione dei risultati mediante conversione da PCS distribuito a catene di Freeman.

Le elaborazioni logiche relative al punto (2) non necessitano di alcun approfondimento: ogni nodo P_i della rete opera sulle striscie di propria pertinenza. Per cio'


P_{p-1}
C
P_2
D
P_1
P_0

che riguarda le operazioni scalari e' opportuno distinguere:

(a) la verifica dell'appartenenza di un punto ad una regione puo' essere effettuata mediante invio di messaggio di verifica dal nodo *C* al nodo P_i cui appartiene l'ordinata *Y* del punto in esame. Il nodo P_i invia a *D* la risposta;

(b) ogni nodo P_i del sistema provvede al computo dell'area delle regioni in esso rappresentate. Il nodo *D* provvede al calcolo finale sommando i diversi contributi;

(c) per quanto riguarda il perimetro, come vedremo nel successivo paragrafo 3.4, ogni nodo P_i puo' agevolmente calcolare la lunghezza dei contorni delle regioni di sua competenza semplicemente innescando una procedura simile a quella di conversione da PCS a catene che tenga conto degli spostamenti effettuati, senza generare link. Ogni contributo viene inviato dal nodo P_i al nodo *D*.

I punti (1) e (3) precedenti richiedono, al contrario, particolare attenzione al fine di ottenere il massimo parallelelismo operando su entita' di natura sequenziale.

3.3 Conversione da catene di Freeman a PCS distribuito

Il nodo *C* esamina sequenzialmente le catene della mappa da convertire. Sulla base della ordinata Y_0 del punto di partenza di ciascuna catena assegna la catena stessa al nodo P_i per cui sia verificata la condizione:

$$YMIN_i \leq Y_0 \leq YMAX_i$$

Il messaggio inviato dal nodo *C* al nodo P_i contiene il comando di conversione *Chain_to_PCS* e la catena da convertire:

$$Send\_Msg(P_i, Chain\_to\_PCS, M, X_0, Y_0, nl, L[0 \div nl-1])$$

Il generico nodo P_i provvede alla conversione sequenziale nello schema PCS della catena ricevuta. Quando la scansione della catena porta alla generazione di un lato di sottostriscia con ordinata esterna all'intervallo di pertinenza del nodo, P_i provvede ad inviare un messaggio identico a quello precedente al nodo interessato (P_{i+1} oppure P_{i-1}) fornendogli la parte di catena non ancora elaborata.

Lo pseudo codice dell'algoritmo di conversione e' mostrato in Fig. 3.2. La procedura e' riferita al generico nodo P_i della rete.

Nell'ipotesi che le regioni della mappa siano uniformemente distribuite, che il tempo necessario allo

```
procedure Distributed Chain_to_PCS()
    x ← x_0
    y ← y_0
    for k = 0, k = k+1 while k < nl do
        switch(L[k])
            case 0:
                x ← x+1
            case 1:
                Side_Insert(M,x,y,1)
                x ← x+1
                y ← y+1
                if (y > YMAX_i) then
                    lr = nl-k-1
                    if (lr ≠ 0) then
                        Send_Msg(P_i+1,Chain_to_PCS,M,x,y,lr,L[k+1÷nl-1])
                    return
            case 2:
                Side_Insert(M,x,y,2)
                y ← y+1
                if (y > YMAX_i) then
                    lr = nl-k-1
                    if (lr ≠ 0) then
                        Send_Msg(P_i+1,Chain_to_PCS,M,x,y,lr,L[k+1÷nl-1])
                    return
            case 3:
                x ← x-1
                Side_Insert(M,x,y,3)
                y ← y+1
                if (y > YMAX_i) then
                    lr = nl-k-1
                    if (lr ≠ 0) then
                        Send_Msg(P_i+1,Chain_to_PCS,M,x,y,lr,L[k+1÷nl-1])
                    return
            case 4:
                x = x-1
            case 5:
                if (y-1 < YMIN_i) then
                    lr = nl-k
                    Send_Msg(P_i-1,Chain_to_PCS,M,x,y,lr,L[k÷nl-1])
                    return
                else
                    x = x-1
                    y = y-1
                    Side_Insert(M,x,y,1)
            case 6:
                if (y-1 < YMIN_i) then
                    lr = nl-k
```

```
                    Send_Msg(P_{i-1},Chain_to_PCS,M,x,y,lr,L[k÷nl-1])
                    return
                else
                    y = y-1
                    Side_Insert(M,x,y,2)
            case 7:
                if (y-1 < YMIN_i) then
                    lr = nl-k
                    Send_Msg(P_{i-1},Chain_to_PCS,M,x,y,lr,L[k÷nl-1])
                    return
                else
                    y = y-1
                    Side_Insert(M,x,y,3)
                    x = x+1
end procedure
```

Fig. 3.2

scambio dei messaggi sia trascurabile e che il semplice controllo operato dal nodo *C* sulle catene sia computazionalmente ininfluente rispetto al processo di conversione, se ne puo' dedurre che il grado di parallelismo raggiungibile con questa soluzione e' direttamente proporzionale al numero *p* di nodi della rete.

3.4 Conversione da PCS distribuito a catene di Freeman

Ogni nodo P_i gestisce ed elabora due tipi di regioni: regioni (e/o buchi) che sono completamente contenute nello spazio $[YMIN_i, YMAX_i]$ di operativita' del nodo e regioni che "spannano" su due o piu' nodi. L'algoritmo che abbiamo definito tiene conto di queste condizioni al fine di massimizzare il grado di parallelismo del sistema.

Ogni nodo P_i ricostruisce dapprima le catene che, con altissima probabilita', fanno parte di regioni distribuite su piu' nodi. Queste catene aperte (con estremi, cioe', non coincidenti) sono inviate al nodo collettore *D* che provvede al loro opportuno riaggancio e visualizzazione. Terminata la ricostruzione delle catene aperte ogni nodo P_i si dedica alla ricostruzione delle (eventuali) catene (chiuse) completamente interne al nodo.

Come vedremo in maggiore dettaglio il nodo *D* non necessita di modificare il verso di percorrenza delle catene aperte ricevute ma soltanto di collegarne gli estremi. Come nel caso sequenziale i contorni delle regioni sono restituiti in senso orario, le frontiere dei buchi in senso antiorario.

La rete di interconnesione tra i nodi *P* deve consentire al generico nodo P_i di richiedere ed ottenere informazioni dal nodo immediatamente superiore o inferiore. Infatti si puo "camminare " da una sottostriscia di ordinata *Y* ad una di identica ordinata soltanto se esiste una sottostriscia di ordinata *Y+1* o *Y-1* che le connette; quando risultasse $Y = YMIN_i$ oppure $Y = YMAX_i$ si renderebbe necessaria una richiesta di informazioni al nodo P_{i-1} oppure P_{i+1}.

Il codice di Fig. 3.3 e' relativo all'algoritmo di conversione da PCS distribuito a catene di Freeman. E' opportuno sottolineare come, per mantenere il corretto verso di percorrenza delle catene, la procedura *PCS_to_Chain* distribuita inneschi dapprima *Bottom_Up* sui lati ovest delle sottostriscie di ordinata $YMIN_i$, quindi *Top_Down* sui lati est delle sottostriscie di ordinata $YMAX_i$ ed infine il normale processo di conversione per le regioni ed i buchi completamente interni. Le catene restituite dalle procedure

```
procedure Distributed_PCS_to_Chain()
  for r ← 0, r ← r+1 while r < Ss_Num(YMIN_i) do
    if (Check_Mark(S_r^{YMIN_i},w) = False) then
      x_b ← Absc(S_r^{YMIN_i},sw)
      y_b ← YMIN_i
      nl = 0
      Bottom_Up(S_r^{YMIN_i})
      if ((x_b = x_e) and (y_b = y_e)) then
        [Send_Msg(D,Display,x_b,y_b,nl,L[0÷nl-1])
      else
        [Send_Msg(D,Close,x_b,y_b,x_e,y_e,nl,L[0÷nl-1])
  for r ← 0, r ← r+1 while r < Ss_Num(YMAX_i) do
    if (Check_Mark(S_r^{YMAX_i},e) = False) then
      x_b ← Absc(S_r^{YMAX_i},ne)
      y_b ← YMAX_i+1
      nl = 0
      Top_Down(S_r^{YMAX_i})
      if ((x_b = x_e) and (y_b = y_e)) then
        [Send_Msg(D,Display,x_b,y_b,nl,L[0÷nl-1])
      else
        [Send_Msg(D,Close,x_b,y_b,x_e,y_e,nl,L[0÷nl-1])
  for y ← YMIN_i+1, y ← y+1 while y < YMAX_i-1 do
    for r ← 0, r ← r+1 while r < Ss_Num(y) do
      if (Check_Mark(S_r^y,w) = False) then
        x_0 ← Absc(S_r^y,sw)
        y_0 ← y
        nl = 0
        Bottom_Up(S_r^y)
        Send_Msg(D,Display,x_0,y_0,nl,L[0÷nl-1])
      else if (Check_Mark(S_r^y,e) = False) then
        x_0 ← Absc(S_r^y,ne)
        y_0 ← y+1
        nl = 0
        Top_Down(S_r^y)
        Send_Msg(D,Display,x_0,y_0,nl,L[0÷nl-1])
end procedure
```

Fig. 3.3

Bottom_Up e *Top_Down* sono inviate al processore *D* con il comando operativo *Close* nel caso di poligonali aperte, con il comando *Display* altrimenti.

La procedura *Bottom_Up* nel caso distribuito (Fig. 3.4) e' sostanzialmente identica alla versione sequenziale quando ci si muova su striscie interne al nodo P_i. Se, nel movimento dal basso verso l'alto, si deve verificare l'esistenza di una sottostriscia superiormente connessa a quella in esame ed appartenente al nodo P_{i+1} un messaggio di richiesta informazioni viene inviato al nodo superiore (*Verify*) fornendo le ascisse dei vertici superiori della sottostriscia .

Essendo richiesta una verifica che comporta l'accesso in sola lettura alla struttura dati PCS del nodo P_{i+1} la risposta (*Reply*) puo' essere inviata senza dover attendere il completamento delle eventuali operazioni in corso. A differenza del caso sequenziale, quando si sia verificata la continuazione della catena nel nodo superiore, la procedura, dopo aver prodotto eventuali link di raccordo, ha termine restituendo a *PCS_to_Chain* la catena ed il punto finale (in coordinate grigliato) raggiunto.

Discorso simmetrico puo essere fatto per la procedura *Top_Down* (Fig. 3.5).

Il nodo *D* della nostra architettura puo' agevolmente procedere alla chiusura delle curve ricevute dai nodi *P* in quanto ogni catena presenta il corretto verso di percorrenza e solo un controllo degli estremi si rende necessario. L'operazione di concatenamento puo' essere condotta in parallelo alla ricostruzione delle curve chiuse interne ad ogni nodo P_i in quanto, come visto, il nodo *D* riceve prima le curve aperte e quindi quelle chiuse.

Quest'ultima considerazione, unitamente al fatto che le richieste di informazioni del generico nodo P_i ai nodi P_{i-1} oppure P_{i+1} sono computazionalmente ininfluenti rispetto al processo di conversione, ci consente di concludere che, anche nel processo di conversione da PCS distribuito a catene di Freeman, si puo' raggiungere un grado di parallelismo proporzionale al numero di nodi del sistema parallelo.

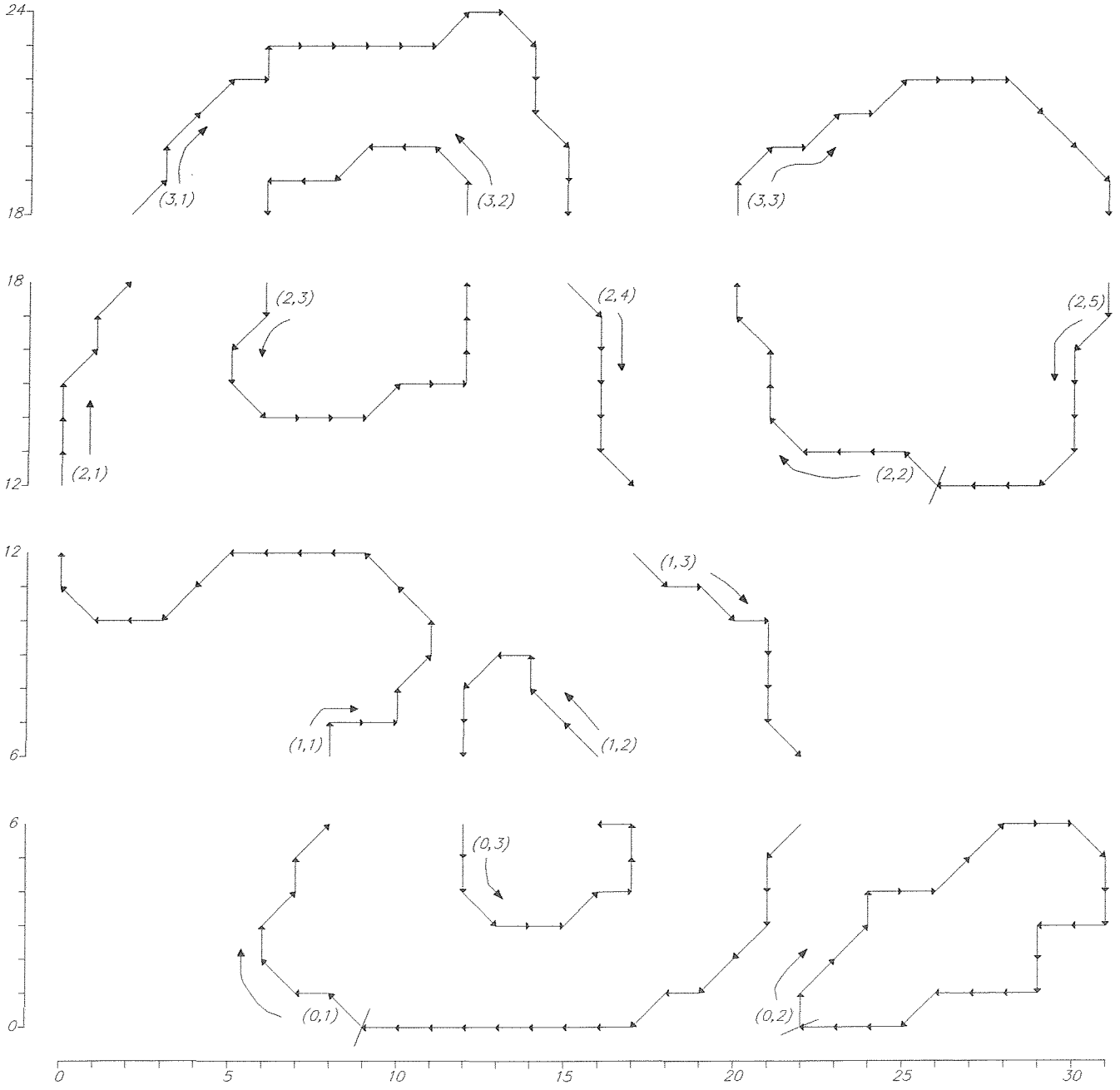
La Fig. 3.6 mostra le catene restituite dai singoli nodi P_i nell'ipotesi che la mappa in esame sia quella di Fig. 2.1 e che il numero di nodi *P* del sistema sia 4. Per ogni catena la figura mostra (in parentesi) il nodo P_i da cui e' stata ricostruita e l'ordine di generazione.

```
procedure Distributed_Bottom_Up (S_r^y)
  if (Check_Mark(S_r^y, w) = True) then
    X_e ← Absc(S_r^y, sw)
    Y_e ← y
    return
  else
    L[nl] ← Type(S_r^y, w)
    nl ← nl+1
    Mark(S_r^y, w)
    if (y < YMAX_i) then
      for k ← 0, k ← k+1 while k < Ss_Num(y+1) do
        if ((Absc(S_r^y, nw) < Absc(S_k^{y+1}, se)) and
                        (Absc(S_r^y, ne) > Absc(S_k^{y+1}, sw))) then
          X_s ← Absc(S_k^{y+1}, sw)
          if (Absc(S_r^y, nw) ≤ X_s) then
            for l ← Absc(S_r^y, nw), l ← l+1 while l < X_s do
              L[nl] ← 0
              nl ← nl+1
            Distributed_Bottom_Up(S_k^{y+1})
            return
          else if ((r = 0) or (Absc(S_{r-1}^y, ne) ≤ X_s)) then
            for l ← X_s, l ← l+1 while l < Absc(S_r^y, nw) do
              L[nl] ← 4
              nl ← nl+1
            Distributed_Bottom_Up(S_k^{y+1})
            return
          else
            for l ← Absc(S_{r-1}^y, ne), l ← l+1 while l < Absc(S_r^y, nw)
              L[nl] ← 4
              nl ← nl+1
            Distributed_Top_Down(S_{r-1}^y)
            return
        for l ← Absc(S_r^y, nw), l ← l+1 while l < Absc(S_r^y, ne) do
          L[nl] ← 0
          nl ← nl+1
        Distributed_Top_Down(S_r^y)
        return
      else
```

```
Send_Msg(P_{i+1}, Verify, y+1, Absc(S_r^y, nw), Absc(S_r^y, ne))
Receive_Msg(P_{i+1}, Reply, Flag, X_s)
if (Flag = True) then
    if (Absc(S_r^y, nw) = X_s) then
        X_e ← Absc(S_r^y, nw)
        Y_e ← y+1
        return
    else if (Absc(S_r^y, nw) < X_s) then
        for l ← Absc(S_r^y, nw), l ← l+1 while l < X_s do
            L[nl] ← 0
            nl ← nl+1
        X_e ← X_s
        Y_e ← y+1
        return
    else if ((r = 0) or (Absc(S_{r-1}^y, ne) ≤ X_s)) then
        for l ← X_s, l ← l+1 while l < Absc(S_r^y, nw) do
            L[nl] ← 4
            nl ← nl+1
        X_e ← X_s
        Y_e ← y+1
        return
    else
        for l ← Absc(S_{r-1}^y, ne), l ← l+1 while l < Absc(S_r^y, nw) do
            L[nl] ← 4
            nl ← nl+1
        Distributed_Top_Down(S_{r-1}^y)
        return
else
    for l ← Absc(S_r^y, nw), l ← l+1 while l < Absc(S_r^y, ne) do
        L[nl] ← 0
        nl ← nl+1
    Distributed_Top_Down(S_r^y)
    return
end procedure
```

Fig. 3.4

```
procedure Distributed_Top_Down (S_r^y)
    if (Check_Mark(S_r^y, e) = True) then
        X_e ← Absc(S_r^y, ne)
        Y_e ← y+1
        return
    else
        L[nl] ← Type(S_r^y, e) + 4
        nl ← nl+1
        Mark(S_r^y, e)
        if (y > YMIN_i) then
            for k ← Ss_Num(y-1)-1, k ← k-1 while k ≥ 0 do
                if ((Absc(S_r^y, sw) < Absc(S_k^{y-1}, ne)) and
                                    (Absc(S_r^y, se) > Absc(S_k^{y-1}, nw))) then
                    X_s ← Absc(S_k^{y-1}, ne)
                    if (Absc(S_r^y, ne) ≥ X_s) then
                        for l ← X_s, l ← l+1 while l < Absc(S_r^y, se) do
                            L[nl] ← 4
                            nl ← nl+1
                        Distributed_Top_Down(S_k^{y-1})
                        return
                    else if ((r = Ss_Num(y)-1) or (Absc(S_{r+1}^y, sw) ≥ X_s)) the
                        for l ← Absc(S_r^y, se), l ← l+1 while l < X_s do
                            L[nl] ← 0
                            nl ← nl+1
                        Distributed_Top_Down(S_k^{y-1})
                        return
                    else
                        for l ← Absc(S_r^y, se), l ← l+1 while l < Absc(S_{r+1}^y, se)
                            L[nl] ← 0
                            nl ← nl+1
                        Distributed_Bottom_Up(S_{r+1}^y)
                        return
            for l ← Absc(S_r^y, sw), l ← l+1 while l < Absc(S_r^y, se) do
                L[nl] ← 4
                nl ← nl+1
            Distributed_Bottom_Up(S_r^y)
            return
        else
```


24
18
(3,1)
(3,2)
(3,3)
18
12
(2,1)
(2,3)
(2,4)
(2,5)
(2,2)
12
6
(1,1)
(1,2)
(1,3)
6
0
(0,1)
(0,3)
(0,2)
0
5
10
15
20
25
30

4. Implementazione del sistema distribuito

Per implementare il sistema di gestione ed elaborazione di mappe in codifica PCS descritto nei paragrafi precedenti si e' fatto uso di una architettura parallela composta di Transputer [INMO87a], [INMO87b], [INMO88a], [INMO88b] connessi da un doppio anello, che permette il fluire di messaggi in entrambe le direzioni. Gli algoritmi distribuiti sono programmati in Occam, facendo uso di codice scritto in linguaggio C per implementare le operazioni gia' definite nell'approccio sequenziale.

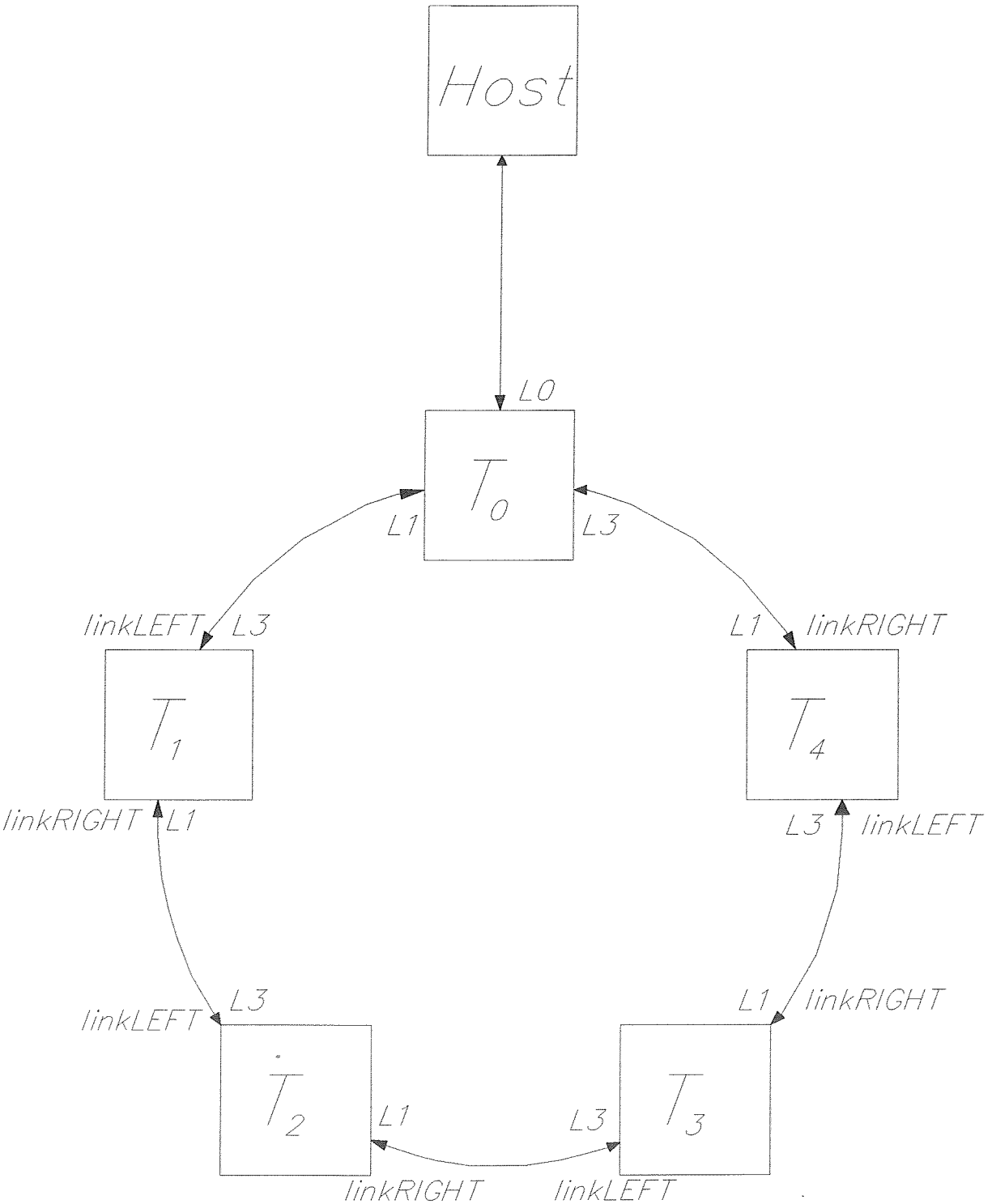
4.1 L'architettura di Transputer

Il sistema parallelo astratto, definito per supportare l'ambiente di elaborazione PCS, puo' essere agevolmente implementato con una rete di Transputer connessi ad anello (fig. 4.1). La connessione per mezzo di link bidirezionali consente di stabilire un doppio anello, su cui scorrono messaggi in entrambe le direzioni. Uno dei Transputer, indicato con T_0 e considerato convenzionalmente il punto di inizio dell'elaborazione distribuita, implementa i nodi *C* e *D* della soluzione distribuita descritta in precedenza, e interfaccia il sistema con l'esterno, tipicamente costituito da un calcolatore (Host) che supporta l'interfaccia grafica e il sistema di archiviazione.

Nell'attuale configurazione sono presenti cinque Transputer (identificati come T_0, T_1, T_2, T_3 e T_4). Il sistema Host e' costituito da un personal computer.

La struttura ad anello si rivela indicata allo scopo, poiche' le comunicazioni previste sono principalmente del tipo per diffusione (inviate a tutti i nodi connessi), oppure avvengono tra i vari nodi dell'architettura sempre tra nodi adiacenti, a cui appartengono insiemi di strisce contigue. I messaggi circolanti sull'anello contengono dati da elaborare o informazioni di controllo relative alle operazioni da compiere.

Le attivita' di instradamento dei messaggi sono molto semplici, consistendo solo nella decisione di far proseguire sull'anello eventuali messaggi non destinati al nodo che compie l'instradamento. La direzione di scorrimento sull'anello viene decisa inizialmente da T_0; durante l'elaborazione nei vari nodi, i messaggi generati possono essere destinati al nodo precedente o a quello seguente nell'anello.


Host
L0
T0
L1
L3
linkLEFT
L3
L1
linkRIGHT
T1
T4
linkRIGHT
L1
L3
linkLEFT
L3
linkLEFT
L1
linkRIGHT
T2
T3
L1
L3
linkRIGHT
linkLEFT

Il nodo T_0 funge da controllore del sistema: in esso sono attivate le funzioni di interfaccia utente, supportata dalle funzioni per l'interfacciamento con video e tastiera e comprendente l'interprete dei comandi relativi alle operazioni da compiere, e l'interfaccia verso il file system per l'acquisizione e la restituzione dei dati.

Tutti gli altri nodi svolgono identiche funzioni:

- conversione da catene di Freeman a PCS e memorizzazione in proprie strutture dati interne; inoltro al nodo successivo delle (porzioni di) catene non pertinenti al nodo stesso;
- elaborazione dei comandi di operazione logica o scalare;
- conversione da PCS a catene di Freeman e inoltro delle (porzioni di) catene generate verso il nodo T_0.

Le attivita' dei vari nodi si svolgono in parallelo: il nodo T_0 invia sull'anello in entrambe le direzioni i messaggi per diffusione che richiedono le operazioni da effettuare; i messaggi per diffusione compiono quindi mediamente solo mezzo giro nell'anello.

I messaggi indirizzati ad un particolare nodo T_i vengono inviati direttamente. La comunicazione avviene in modo da minimizzare il percorso tra T_0 e il nodo considerato, pertanto il messaggio viene inviato sull'anello nella direzione opportuna: ad esempio la comunicazione da T_0 a T_1 o a T_4 avviene direttamente, mentre la comunicazione da T_0 a T_2 avviene attraverso T_1 e quella da T_0 a T_3 avviene attraverso T_4. Nel nodo T_0 deve essere prevista una struttura dati contenente, per ogni nodo T_i connesso sull'anello, gli estremi $[YMIN_i, YMAX_i]$ delle ordinate relative alle strisce PCS contenute nel nodo stesso e la direzione in cui inviare i messaggi al nodo T_i. La ripartizione delle strisce PCS ai nodi e' statica; in caso di ripartizione dinamica la struttura dati contenuta in T_0 sara' convenientemente aggiornata.

I messaggi relativi alle operazioni di conversione sono costituiti da porzioni di catene di Freeman circolanti tra i vari nodi del sistema. Nella conversione da rappresentazione con catene di Freeman a PCS, il nodo T_0 invia la catena da convertire al nodo che contiene la striscia relativa al punto di inizio. Nella conversione da PCS a catene di

Freeman tutte le (porzioni di) catene generate vengono inviate al nodo T_0 scorrendo l'anello nella direzione che consente il tragitto piu' breve.

Le operazioni logiche o scalari su mappe non richiedono generalmente comunicazioni tra i nodi. Fa eccezione il calcolo dell'area di una mappa; dopo che ogni nodo ha calcolato localmente l'area della porzione di mappa in esso contenuta (eventualmente 0), comunica il risultato al nodo T_0, sul tragitto piu' breve. Il calcolo del perimetro viene effettuato con procedura analoga a quella di conversione da PCS a catene; dopo che ogni nodo ha calcolato localmente il valore (eventualmente 0), comunica il risultato al nodo T_0, sul tragitto piu' breve.

4.2 Schema dei processi di implementazione

Vengono illustrati in questo paragrafo i processi Occam e C che implementano gli algoritmi descritti nel Cap. 3, mettendo in evidenza il formato dei messaggi circolanti nel sistema e le strutture dati utilizzate.

4.2.1 I processi attivati sul nodo T_0

Un processo CONTROLLER e' attivo sempre, con il compito di ricevere da terminale le richieste di operazione che possono essere:

Initialize
inizializzazione, effettuata una sola volta in una fase di avviamento del sistema. Le informazioni di inizializzazione relative alla configurazione del sistema vengono memorizzate in una struttura dati che contiene, per ogni nodo della rete, la direzione (oraria, antioraria) per l'instradamento dei messaggi. Vengono quindi specificati gli estremi delle coordinate x e y da trattare. Il sistema calcola automaticamente gli estremi delle ordinate per le strisce contenute nei vari nodi, li visualizza all'utente per consentire eventuali variazioni manuali, quindi li memorizza e provvede a comunicarli con un messaggio *(Initialize)* ad ogni nodo, che li conserva per le successive elaborazioni. In ogni nodo T_i e' presente una copia delle informazioni corrispondenti al nodo stesso; la direzione per l'instradamento dei messaggi ha senso opposto a quella memorizzata in T_0, poiche' indica l'instradamento dei messaggi dal nodo T_i verso il nodo T_0.

Chain_to_PCS
acquisizione dei dati e conversione in formato interno; sono specificati il nome del file contenente i dati e un

nome simbolico *M* (identificatore interno della mappa). Si innesca una serie di operazioni di lettura richieste al file system; le coordinate iniziali della catena di Freeman da convertire consentono di identificare il nodo T_i a cui e' demandata la conversione. Al nodo T_i viene inviata la catena da convertire, con una successione di messaggi contenenti i dati. Il primo messaggio:

$$(Chain\_to\_PCS, M, X_0, Y_0, nl)$$

inviato nell'opportuna direzione, stabilisce il percorso su cui vengono instradati tutti i messaggi relativi alla catena, e il loro numero. I messaggi successivi sono costituiti dai singoli link della catena *L[0], L[1], ..., L[nl-1]*.

I nodi compresi tra T_0 e il nodo T_i cosi' individuato considerano tali messaggi in transito, e non ne effettuano alcuna elaborazione.

Iniziata l'elaborazione della catena da convertire in forma PCS, essa prosegue sullo stesso nodo T_i fino a quando un punto *X, Y* trattato risulta non compreso nello spazio di ordinate $[YMIN_i, YMAX_i]$ proprio del nodo. Quando cio' si verifica, T_i genera un messaggio *Chain_to_PCS* per un nodo T_j contiguo ($j = i+1$ oppure $j = i-1$), e ulteriori dati ricevuti sono considerati in transito e inoltrati al nodo opportuno. In tal modo ogni nodo controlla ad ogni istante l'elaborazione di propria pertinenza, ottenendo cosi' una distribuzione uniforme del carico di lavoro sui vari nodi della rete. Si noti che il transito dei messaggi non richiede funzioni (eventualmente complesse) di instradamento; puo' essere peraltro necessaria la bufferizzazione dei messaggi, per migliorare l'efficienza delle comunicazioni.

Le comunicazioni sull'anello sono effettuate byte a byte in parallelo a quelle di lettura da file system per aumentare le prestazioni del sistema. Si stabilisce di conseguenza un pipeline costituito dal processo file system, da CONTROLLER e dai processi CONTROL_COM attraversati dai messaggi sui vari nodi. L'attivita' di CONTROLLER prosegue con altre elaborazioni quando l'ultimo messaggio contenente i dati viene ricevuto dal primo nodo del pipeline.

Union, Intersection, Difference e Complement

richiesta di operazione di unione, intersezione, differenza su piu' mappe o complemento di una mappa; vengono specificati i nomi simbolici delle mappe su cui operare, e il nome simbolico della mappa risultante. A tutti i nodi della rete sono inviati per diffusione i messaggi (*Union, Intersection, Difference, Complement*) che richiedono l'operazione. Per uniformita' i processi di comunicazione

sui vari nodi sono tutti identici e non viene identificato un primo (o ultimo) nodo sull'anello. La comunicazione per diffusione, pertanto, termina quando il messaggio (*Union, Intersection, Difference, Complement*) perviene due volte allo stesso nodo, con provenienza dalle due opposte direzioni dell'anello.

Il processo CONTROLLER continua l'attivita' allorche' il messaggio per diffusione e' stato inviato sull'anello.

Point_Inclusion, Area, Boundary_Length
operazioni scalari sulla mappa individuata dal nome *M* specificato; il messaggio *Point_Inclusion* e' inviato al nodo T_i che contiene le coordinate *X, Y* specificate e se ne attende la risposta. I messaggi *Area* e *Boundary_Length* sono inviati per diffusione; le risposte provenienti da tutti i nodi T_i vengono contate e i corrispondenti valori sono sommati tra loro per calcolare il risultato.

PCS_to_Chain
restituzione dei dati dopo la conversione in forma di catene di Freeman; viene specificato il nome simbolico della mappa e il nome del file destinato a contenere i dati risultanti (se ne e' richiesta la memorizzazione). A tutti i nodi della rete viene inviato per diffusione il messaggio (*PCS_to_Chain*) che richiede l'operazione. Si attendono dai vari nodi T_i le catene complete e quelle aperte, per le quali si procede alla ricostruzione ("agganciando" tratti adiacenti).

End
fine lavoro; innesca le comunicazioni necessarie per la terminazione delle attivita'.

I messaggi generati in corrispondenza delle operazioni sopra elencate hanno il seguente formato:

Init:- ident. T_i *dest., XMIN, YMIN, XMAX, YMAX;*

Chain_to_PCS :- ident. T_i *dest., M,* X_0,Y_0,*nl;*

Link :- L[i];

Union :- M_1, M_2, M_{ris};

Intersection :- M_1, M_2, M_{ris};

Difference :- M_1, M_2, M_{ris};

Complement :- M_1, M_{ris};

Point_Inclusion :- ident. T_i *dest., M, X, Y;*

Point_Answer :- M, T/F;

Area :- M;

Area_Value :- M, A_{ris};

Boundary_Length :- M;

BL_Value :- M, BL_{ris};

PCS_to_Chain :- M;

Verify :- M, Y, X_b, X_e;

Reply :- M, T/F, X_s;

Close :- M, X_s, Y_s, X_e, Y_e, nl, L[0], ..., L[nl-1];

Display :- M, X_0, Y_0, nl, L[0], ..., L[nl-1];

End.

4.2.2 I processi attivati sui vari nodi della rete.

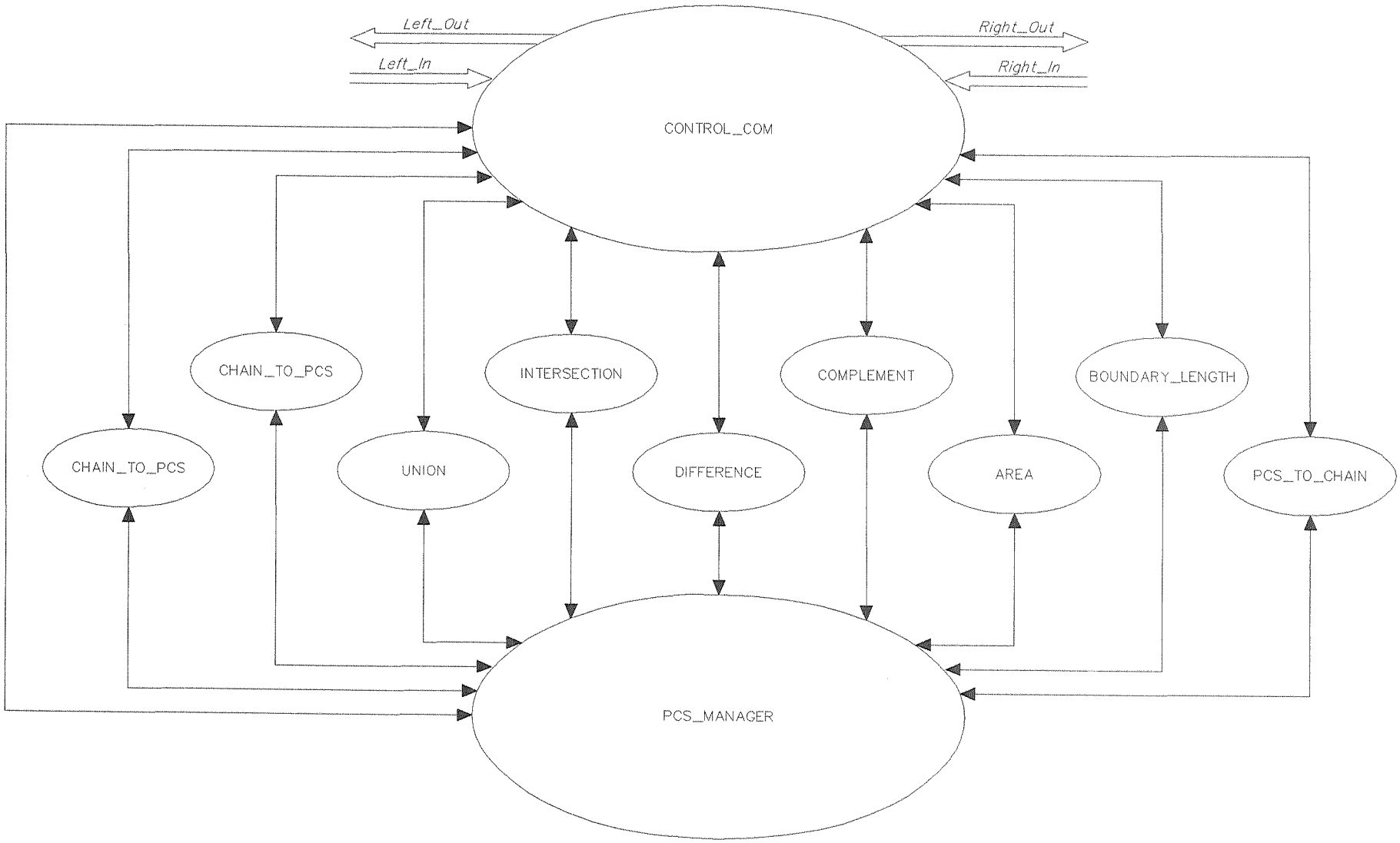
Le attivita' eseguite sui singoli nodi comprendono funzioni di elaborazione su strutture PCS e funzioni di comunicazione per la gestione del sistema. I processi Occam allocati su ogni nodo sono permanentemente attivi. Essi comprendono un processo di comunicazione e controllo (CONTROL_COM) e un processo di elaborazione, composto dai processi, programmati in linguaggio C e attivati in parallelo, che eseguono le funzioni richieste: UNION, INTERSECTION, DIFFERENCE, COMPLEMENT, AREA, BOUNDARY_LENGTH, CHAIN_TO_PCS, PCS_TO_CHAIN. La struttura dati che contiene la rappresentazione PCS e' incapsulata in un processo gestore PCS_MANAGER (*Fig.* 4.2).

Per la funzione *Point_Inclusion*, cosi' come per le richieste di informazioni da e verso altri nodi (*Verify, Reply*), non sono previsti processi distinti; il processo CONTROL_COM interagisce direttamente con PCS_MANAGER per quanto riguarda tali richieste.

Come spiegato nel seguito, il processo CHAIN_TO_PCS e' duplicato, per poter agire simultaneamente su catene provenienti da entrambe le direzioni sull'anello.

Il processo CONTROL_COM gestisce le comunicazioni, attendendo simultaneamente sui due link esterni (*Right_In, Left_In*) i messaggi circolanti sull'anello nelle due direzioni, tenendo conto del possibile stato dell'elaborazione in corso sul nodo. Gli stati ammessi sono (*Fig.* 4.3):

INITIAL:
quando non e' avvenuta alcuna comunicazione; si attende il messaggio *Initialize*. Ogni altro messaggio e' considerato in transito, e viene quindi inoltrato al nodo successivo sul link opportuno (i messaggi provenienti da *Right_In* sono inviati a *Left_Out*, e quelli provenienti da *Left_In* a *Right_Out*). Dopo aver ricevuto *Initialize*, ed aver memorizzato la direzione di invio dei messaggi a T_0, e gli estremi $XMIN_i$, $YMIN_i$, $XMAX_i$, $YMAX_i$, si effettua una transizione nello stato *OPERATING*, preparandosi a ricevere


Left_Out
Left_In
Right_Out
Right_In
CONTROL_COM
CHAIN_TO_PCS
INTERSECTION
COMPLEMENT
BOUNDARY_LENGTH
CHAIN_TO_PCS
UNION
DIFFERENCE
AREA
PCS_TO_CHAIN
PCS_MANAGER

richieste di elaborazione.

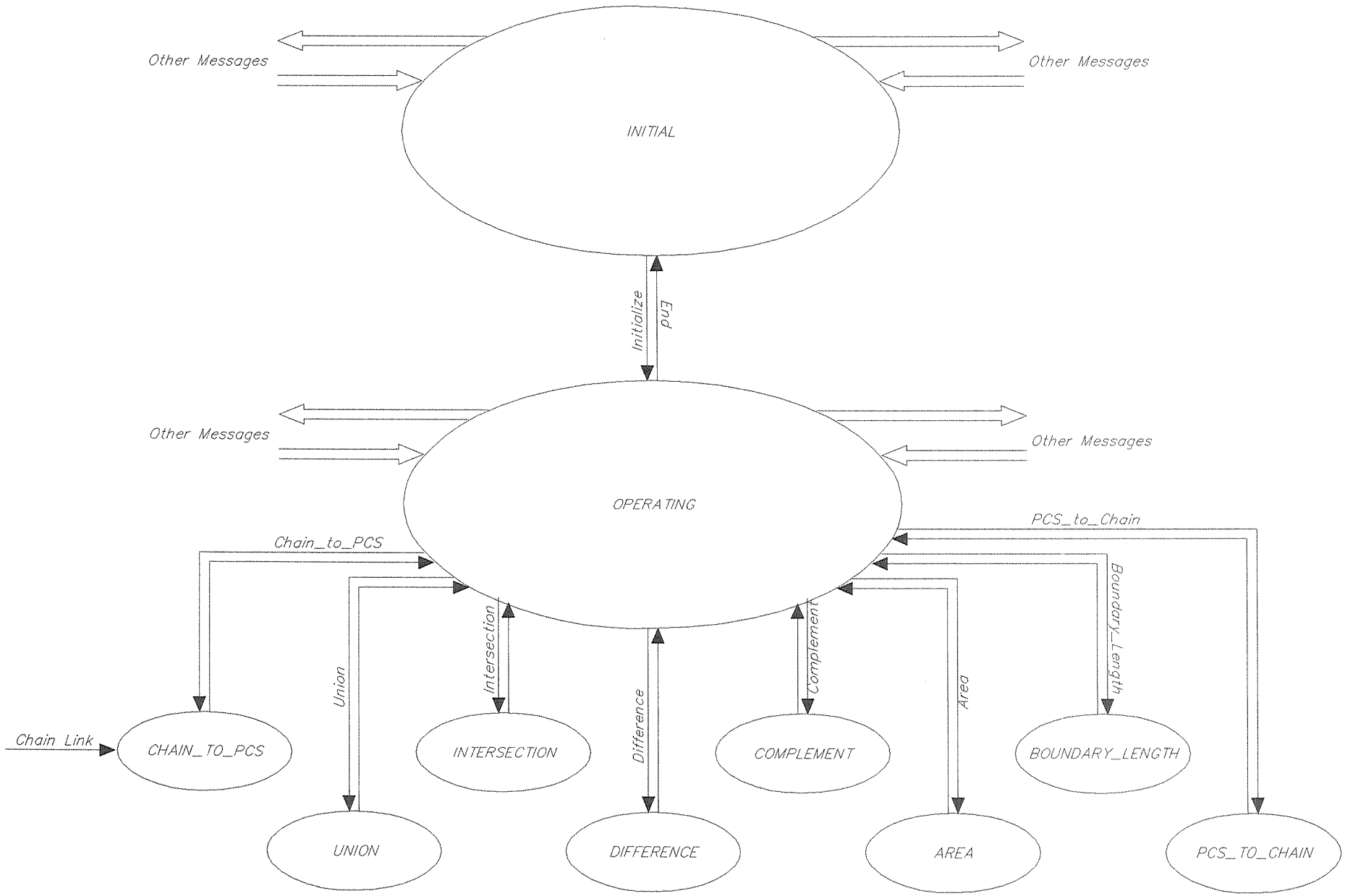
OPERATING:
e' lo stato in cui si ricevono le richieste di effettuare operazioni: *Chain_to_PCS, Union, Intersection, Difference, Complement, Area, Boundary_Length, PCS_to_Chain*. Per ognuna di esse si attiva un processo di elaborazione, e si esegue la corrispondente transizione di stato (*CHAIN_TO_PCS, UNION, INTERSECTION, DIFFERENCE, COMPLEMENT, AREA, BOUNDARY_LENGTH, PCS_TO_CHAIN*). La richiesta viene inviata al nodo successivo, nel caso si tratti di una comunicazione per diffusione, fino a quando non si raggiunge un nodo in cui l'indicatore di stato segnala gia' il valore corrispondente.

Il messaggio di terminazione (*End*) provoca la transizione nello stato *INITIAL*.

CHAIN_TO_PCS:
e' lo stato corrispondente alla ricezione di un messaggio *Chain_to_PCS*. Le coordinate X_0, Y_0 vengono comunicate al processo CHAIN_TO_PCS, mentre il numero intero *nl* viene usato come contatore per controllare i dati successivi.

Poiche' possono pervenire al nodo richieste di conversione da entrambe le direzioni sull'anello (cio' e' vero in particolare per richieste di conversione su sottocatene, provenienti da altri nodi che hanno gia' effettuato la conversione sulla porzione di catena di rispettiva pertinenza), il processo CHAIN_TO_PCS e' duplicato, in modo da garantire il massimo parallelismo di elaborazione, corrispondente alla simultanea conversione di due catene. Le richieste provenienti in direzione oraria (da *Right_In*) sono comunicate dal processo CONTROL_COM a CHAIN_TO_PCS attraverso il canale CONVIN.RI, mentre quelle provenienti in direzione antioraria (da *Left_In*) sono comunicate su CONVIN.LI. Due indicatori booleani (CONVRIGHT, CONVLEFT) permettono di identificare le operazioni di conversione avviate, e di controllarne la corretta sequenzializzazione. Due indicatori booleani (TRANSLEFT, TRANSRIGHT) permettono inoltre di ricordare su quale canale sono stati inviati al nodo successivo i dati della catena considerati in transito, poiche' per garantire la correttezza e' ammessa la circolazione di una sola sottocatena in transito in una stessa direzione.

I dati provenienti da *Right_In* e *Left_In* vengono inoltrati in accordo con gli indicatori CONVRIGHT e CONVLEFT. Per le operazioni gia' attivate si provvede alla trasmissione dei dati in arrivo sul corrispondente canale CONVIN.LI oppure CONVIN.RI, aggiornando il relativo contatore; se si presenta una richiesta relativa ad un processo CHAIN_TO_PCS ancora libero, esso viene attivato come gia' descritto in precedenza.


Other Messages
Other Messages
INITIAL
Initialize
End
Other Messages
Other Messages
OPERATING
Chain_to_PCS
PCS_to_Chain
Union
Intersection
Difference
Complement
Area
Boundary_Length
Chain Link
CHAIN_TO_PCS
INTERSECTION
COMPLEMENT
BOUNDARY_LENGTH
UNION
DIFFERENCE
AREA
PCS_TO_CHAIN

Il processo CONTROL_COM attende da CHAIN_TO_PCS l'eventuale segnalazione di aver raggiunto nella catena un punto esterno allo spazio controllato dal nodo (un punto con ordinata *Y* esterna all'intervallo fissato per il nodo). In presenza di tale segnalazione viene generato un messaggio *Chain_to_PCS* con punto di partenza *X, Y* calcolato da CHAIN_TO_PCS e *lr* valore residuo del contatore dei link. Il messaggio viene inoltrato sul canale *Left_out* se $Y<YMIN_i$, sul canale *Right_Out* se $Y>YMAX_i$, dopo aver verificato che il valore degli indicatori TRANSLEFT e TRANSRIGHT consenta il transito. Gli indicatori vengono quindi aggiornati opportunamente. Il processo CHAIN_TO_PCS che ha causato la segnalazione viene considerato libero. Se entrambi i processi CHAIN_TO_PCS generano sottocatene da elaborare su altri nodi, la prosecuzione viene autorizzata in funzione del valore di TRANSLEFT e TRANSRIGHT: se l'elaborazione successiva coinvolge nodi distinti, per i quali la comunicazione avviene sui due diversi canali *Left_Out* e *Right_Out*, cioe' nelle due direzioni sull'anello, viene abilitata per entrambi i processi CHAIN_TO_PCS. Se invece la comunicazione relativa ad entrambe le catene deve avvenire sullo stesso canale (l'elaborazione prosegue sullo stesso nodo), solo il primo processo CHAIN_TO_PCS da luogo alla prosecuzione, mentre il secondo rimane in attesa. Gli indicatori TRANSLEFT e TRANSRIGHT consentono di individuare in quale combinazione ci si trovi e di decidere per la prosecuzione in modo corretto.

Il meccanismo di controllo basato sugli indicatori CONVRIGHT, CONVLEFT, TRANSRIGHT, TRANSLEFT, puo' apparire inutilmente macchinoso, ma e' giustificato dal tentativo di conseguire la massima banda di elaborazione possibile con la struttura della rete (due sottocatene in elaborazione contemporaneamente in ogni nodo, e due flussi di sottocatene circolanti sull'anello nelle due direzioni), anche per tenere conto del meccanismo di comunicazione sincrona tra processi Occam.

Le comunicazioni sospese in attesa di ottenere un processo CHAIN_TO_PCS o un canale libero, possono essere bufferizzate nel processo CONTROL_COM per rendere asincrone le attivita' di comunicazione nei vari nodi. In una prima implementazione la soluzione adottata e' stata pero' quella di sfruttare il meccanismo di comunicazione sincrona dei canali Occam, ponendo attenzione ad eventuali situazioni di potenziale dead-lock (comunicazioni tra CONTROL_COM e CHAIN_TO_PCS; messaggi inviati e ricevuti su uno stesso link dell'anello).

Allorche' il contatore *lr* (lunghezza residua della catena da convertire) raggiunge il valore 0, il rispettivo processo CHAIN_TO_PCS viene liberato. Quando entrambi i processi CHAIN_TO_PCS sono liberi, si effettua una

transizione allo stato *OPERATING*.

UNION:
e' lo stato in cui e' attivo il processo UNION.

INTERSECTION:
e' lo stato in cui e' attivo il processo INTERSECTION.

DIFFERENCE:
e' lo stato in cui e' attivo il processo DIFFERENCE.

COMPLEMENT:
e' lo stato in cui e' attivo il processo COMPLEMENT.

AREA:
e' lo stato in cui e' attivo il processo AREA. Il processo CONTROL_COM riceve dal processo AREA il messaggio *Area_Value*, che viene inoltrato verso T_0 sul percorso piu' breve.

BOUNDARY_LENGTH:
e' lo stato in cui e' attivo il processo BOUNDARY_LENGTH. Il processo CONTROL_COM riceve dal processo BOUNDARY_LENGTH il messaggio *BL_Value*, che viene inoltrato verso T_0 sul percorso piu' breve.

PCS_TO_CHAIN:
e' lo stato in cui e' attivo il processo PCS_TO_CHAIN. Possono essere ricevuti e inoltrati messaggi contenenti le catene di Freeman, da inviare al nodo T_0. Il processo PCS_TO_CHAIN seleziona per prime le catene con estremo $Y_0 = YMIN_i$ oppure $Y_0 = YMAX_i$, che con elevata probabilita' sono catene aperte. Tali catene sono comunicate per mezzo del processo CONTROL_COM verso il nodo T_0. Il processo PCS_TO_CHAIN applica quindi l'algoritmo di conversione alle restanti catene e provvede ad inoltrare anche queste al nodo T_0.

In ognuno degli stati sopra specificati, possono pervenire richieste (*Verify, Reply*) di controllo di informazioni contenute nella struttura dati PCS, per determinare la prosecuzione di processi di conversione attivi su nodi contigui. Esse vengono servite immediatamente, comunicando direttamente con il processo PCS_MANAGER. Un analogo trattamento e' riservato al messaggio *Point_Inclusion* per il quale non si procede ad un cambiamento di stato.

5. STRUMENTI DI CONTROLLO E VALUTAZIONE

Gli algoritmi descritti nei paragrafi precedenti presentano, nell'implementazione parallela, oltre alle attivita' di elaborazione anche funzioni di comunicazione, che vanno considerate nel loro complesso per determinare i tempi di esecuzione. La distribuzione del carico di lavoro sui singoli nodi e il traffico sulla struttura di interconnessione possono influire anche pesantemente sulle prestazioni ottenibili.

E' opportuno prevedere nel sistema appositi strumenti per il controllo e la valutazione delle attivita', per poter effettuare le scelte che consentono l'ottimizzazione delle prestazioni del sistema [Mast85, Pomp89].

Per la valutazione dei tempi di esecuzione, elemento di interesse e' l'indice di incremento delle prestazioni della soluzione parallela *(P)* rispetto a quella sequenziale *(S)*. Nel seguito si considera N il numero dei nodi dell'architettura parallela.

5.1 Tools di valutazione

Le comunicazioni necessarie a far circolare i dati sull'anello introducono un tempo di comunicazione *(TC)* proporzionale al tempo di comunicazione t_c sul singolo collegamento tra Transputer successivi. Si puo' considerare il tempo di comunicazione TC_{ij} tra 2 Transputer T_i e T_j (indicato per brevita' con TC_i se uno dei due nodi comunicanti e' il nodo T_0) come il prodotto

$$TC_{ij} = t_c * m * b$$

ove m indica il numero di tratti di collegamento (il numero di Transputer) attraversati, e b e' indicativo del numero di bit trasferiti.

Si calcola *TC* come valore medio di TC_1, TC_2, TC_3 e TC_4 tempi necessari per la comunicazione tra T_0 e il nodo T_1, T_2, T_3 e T_4 rispettivamente. I tempi TC_i sono ridotti, con media $t_c*b*(N+2)/4$ anziche' $t_c b*(N+1)/2$, per effetto del meccanismo di comunicazione sull'anello in entrambe le direzioni. Nelle comunicazioni per diffusione il tempo *TCD* per completare l'operazione e' di $t_c*b*(\frac{N}{2}+1)$ rispetto al tempo $t_c*b*(N+1)$ necessario a compiere un intero giro dell'anello.

Le operazioni logiche su piu' mappe o quelle scalari su una singola mappa, vengono effettuate su strutture partizionate, eseguendo lo stesso algoritmo della soluzione sequenziale. Nell'ipotesi di equidistribuzione dei dati, si

puo' quindi assumere che $P = S/N$. Non si hanno comunicazioni durante l'esecuzione delle operazioni,pertanto si puo' stimare il solo costo della comunicazione per diffusione iniziale. Nel caso di operazioni di calcolo di Area o di Perimetro, si hanno comunicazioni del risultato da ogni nodo T_i al nodo T_0. Il costo di comunicazione complessivo per ogni nodo ($2*TC_i$) puo' essere stimato, nell'architettura considerata, intorno ai 10 μsec. (10 byte trasferiti su linee con velocita' di circa 8 Mbit al sec.).

Le operazioni di conversione da catene di Freeman a rappresentazione PCS *(FPCS)* e viceversa *(PCSF)* richiedono una valutazione accurata. Nella soluzione sequenziale, si puo' assumere un costo di elaborazione complessivo *FPCS* proporzionale al numero delle catene *(C)* moltiplicato per il numero (medio) dei link *(nl)* in ogni catena. Il costo *PCSF* e' proporzionale al numero delle catene *(C)* moltiplicato per il numero (medio) delle strisce *(st)* per ogni catena, tenendo conto di un costo di ricerca delle strisce appartenenti ad una stessa catena.

Nell'elaborazione parallela, se si considerano le catene equidistribuite sulle strisce (sui nodi di elaborazione), si possono stimare i tempi di elaborazione $P=S/N$. Cio' anche nell'ipotesi di elaborare su nodi distinti sottocatene appartenenti alla stessa catena. I tempi di elaborazione sui singoli nodi rimangono proporzionali a $C*nl/N$ e a $C*st/N$.

Al tempo di elaborazione occorre aggiungere un tempo di comunicazione *TC*, nell'ipotesi che le fasi di elaborazione e di comunicazione siano mantenute distinte. Tale ipotesi, relativa al caso peggiore, puo' essere rilasciata nel caso dell'architettura proposta, che permette di effettuare le comunicazioni in parallelo con l'elaborazione.

Se si ritiene verosimile che i processi CONTROL_COM introducano un tempo di transito gia' compreso nel calcolo del tempo di trasmissione sul collegamento fisico, e che i dati acquisiti da un nodo vengano immediatamente elaborati, il costo di comunicazione si riduce al solo tempo t_c per il messaggio *Chain_to_PCS* iniziale e diviene trascurabile al crescere del numero di elementi della catena. Cio' rimane vero anche nel caso in cui la catena venga elaborata in sottocatene da piu' Transputer. In realta' anche l'ipotesi di elaborazione immediata puo' essere rilasciata; sotto l'ipotesi meno restrittiva che non si introducano tempi morti (nodi inattivi perche' le comunicazioni sono ritardate nel transito) o non si crei stallo, ha poca importanza che sia ritardata l'elaborazione di una sottocatena, se sullo stesso nodo e' gia' in elaborazione un'altra sottocatena.

Per una valutazione precisa dei tempi sopra indicati, appare opportuno assumere come unita' di tempo quello

necessario per elaborare un singolo elemento di catena. Se il sistema mette a disposizione un conteggio relativamente al numero di catene e di elementi elaborati su ogni singolo nodo, si puo' avere una stima della equidistribuzione dei dati. Se ad esso si aggiunge il conteggio dei dati in transito su ogni nodo, si puo' stimare l'efficienza della politica di instradamento adottata. Infine, per valutare l'efficienza complessiva del sistema, occorre disporre di un indice del tempo occupato in ogni nodo per elaborazioni rispetto al tempo di attesa (processi CHAIN_TO_PCS, PCS_TO_CHAIN inattivi) ed al tempo di comunicazione (processo CONTROL_COM attivo).

5.2 Bilanciamento di carico

Nell'ipotesi di strisce uniformemente distribuite sui nodi, si puo' dedurre una distribuzione uniforme anche del carico di lavoro. A partire dalle valutazioni, se occorre effettuare un bilanciamento e' possibile cambiare (eventualmente in modo dinamico) gli estremi delle ordinate (il numero di strisce) gestite dai vari nodi.

5.3 Scheduling distribuito

I nodi della rete elaborano le varie funzioni su dati partizionati. L' attivazione di tali funzioni avviene in parallelo sui vari nodi per effetto di una comunicazione per diffusione, e non richiede scelte di scheduling. Per la conversione dei dati dalla forma in catene di Freeman a PCS e viceversa, al contrario, l'ordine con cui le catene sono inviate ai nodi per l'elaborazione puo' influire sui tempi di elaborazione, ovvero puo' causare tempi di inattivita' di alcuni nodi rispetto ad altri, oppure implicare tempi di attesa per la comunicazione, qualora le catene in transito abbiano saturato alcuni tratti del collegamento tra i Transputer. Nella soluzione implementata non e' stata prevista alcuna disciplina per l'instradamento delle catene di Freeman da convertire.

A partire dalle valutazioni sul funzionamento del sistema, puo' essere conveniente realizzare una politica di attivazione delle elaborazioni o di instradamento dei dati in circolazione che permetta di ridurre i tempi di inattivita' dei processi di elaborazione e migliorare il traffico sull'anello.

In particolare il nodo T_0 puo' svolgere funzioni di sequenzializzazione dell'invio in modo da attivare circolarmente tutti i nodi T_i. L'instradamento sui nodi T_i per le catene in transito o inoltrate durante l'elaborazione, puo' avvenire sfruttando convenientemente buffer di dimensioni opportune, gestiti dal processo

CONTROL_COM. Nel caso in cui le funzioni di CONTROL_COM assumano un peso predominante nell'elaborazione sul nodo T_i, si puo' realizzare ogni nodo per mezzo di due Transputer, connessi tra loro, di cui uno dedicato alle funzioni di memorizzazione ed elaborazione dei dati, il secondo dedicato all'esecuzione delle funzioni di CONTROL_COM.

6. Conclusioni

E' stato presentato nei capitoli precedenti un sistema parallelo per la gestione di informazioni spaziali, applicato in particolare all'elaborazione di carte tematiche.

La rappresentazione scelta, denominata PCS [Mont84] si e' rivelata adatta al trattamento con algoritmi paralleli, tali da conseguire, rispetto ad elaborazioni sequenziali, prestazioni proporzionali al numero di processori utilizzati.

Nel caso delle piu' comuni operazioni scalari o logiche, gli algoritmi applicati sono costituiti da piu' copie, eseguite in parallelo su partizioni disgiunte dei dati, dello stesso algoritmo sequenziale, mentre e' stato necessario definire un opportuno algoritmo parallelo ed un adeguato schema di comunicazione per supportare le operazioni di conversione da rappresentazione PCS a catene di Freeman e viceversa.

Sebbene le operazioni di conversione operino su strutture per loro natura strettamente sequenziali, gli algoritmi paralleli definiti hanno consentito di stimare anche per tali operazioni un incremento di prestazioni proporzionale al numero di processori impiegati.

La soluzione descritta e' stata implementata su una rete di Transputer connessi ad anello. L'implementazione utilizza il codice, in linguaggio C, gia' sviluppato in precedenza per gli algoritmi sequenziali, mentre si e' fatto uso del linguaggio Occam per programmare le attivita' concorrenti del sistema.

Attualmente e' in corso di realizzazione l'integrazione delle funzionalita' del sistema con tools di valutazione. Sulla base di esperimenti effettuati si potra' procedere se necessario ad aggiustamenti successivi mediante lo sviluppo di funzioni di scheduling delle attivita' di elaborazione e di comunicazione, con criteri di bilanciamento di carico e di eliminazione di eventuali colli di bottiglia.

Bibliografia

[Arab87] H. R. Arabnia, M. A. Oliver, A Transputer network for the arbitrary rotation of digitized images, *The Computer Journal*, 3, 5, 425-432.

[Arab89] H. R. Arabnia, M. A. Oliver, A Transputer network for fast operations on digitized images, *Computer Graphics Forum*, 8, 1989, 3-11.

[Free74] H. Freeman, Computer processing of line-drawing images, *ACM Computing Surveys*, 6, 1974, 57-97.

[INMO87a] INMOS, Occam2 Toolset: Reference/User Manual, 1987.

[INMO87b] INMOS, 3L Parallel C: Reference/User Manual, 1987.

[INMO88a] INMOS, IMS B008: User Guide and Reference Manual, 1988.

[INMO88a] INMOS, IMS B012: User Guide and Reference Manual, 1988.

[Kane87] R. Kane, S. Sahni, Systolic algorithms for rectilinear polygons, *Computer Aided Design*, 19, 1, 1987, 15-24.

[Kris87] D. Krishnan, L. M. Patnaik, Systolic architecture for boolean operations on polygons and polyhedra, *Computer Graphics Forum*, 6, 1987, 203-210.

[Merr73] R. D. Merrill, Representation of Contours and Regions for Efficient Computer Search, *Communications of ACM* 16, 2, 1973, 69-82.

[Mast85] L. Matropietro, A. Tomasi, The MuTEAM testbed for distributed architectures, *MIMI '85 International Conference on Mini and Microcomputers*, 1985.

[Mont84] C. Montani, Region Representation: Parallel Connected Stripes, *Computer Vision, Graphics, and Image Processing*, 28, 1984, 139-165.

[Pomp89] R. Pompei, M. Sciortino, A. Tomasi, *Un ambiente di sviluppo in Occam per architetture parallele basate su Transputer*, Atti Congresso Annuale AICA, Trieste 4-6 Ottobre 1989.

[Same84] H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys*, 16, 2, 1984,

187-260.

[Same90] H. Samet, *Applications of spatial data structures*, Addison Wesley Publishing Company, Reading (MA), 1990.

[Wu89] A. Y. Wu, S. K. Bhaskar, A. Rosenfeld, Parallel processing of region boundaries, *Pattern Recognition*, 22, 2, 1989, 165-172.