UNIVERSITÀ DEGLI STUDI DI PARMA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Dottorato di Ricerca in Tecnologie dell'Informazione
VII Ciclo

Alberto Broggi

Strumenti e Tecniche per l'Elaborazione di Immagini mediante Architetture Massivamente Parallele

Dissertazione presentata per il conseguimento del titolo di Dottore di Ricerca

Prefazione

Il progetto PAPRICA ebbe inizio nel 1988 con lo scopo di definire un sistema VLSI per l'elaborazione efficiente di strutture dati bidimensionali.

L'obiettivo principale del progetto è lo sviluppo di un sottosistema che possa operare come un coprocessore di un calcolatore tradizionale e, in prospettiva, come un modulo specializzato per sistemi dedicati alla preelaborazione di immagini, emulazione di reti neurali cellulari, e in generale per l'esecuzione di operazioni esprimibili mediante paradigmi cellulari SIMD massivamente paralleli come Morfologia Matematica e Automi Cellulari.

L'architettura PAPRICA è stata ampiamente utilizzata per i filtraggi di base dell'elaborazione di basso livello fino all'implementazione di algoritmi di riconoscimento e inseguimento di caratteristiche, mostrando prestazioni incoraggianti perfino con la prima versione del prototipo.

Il presente lavoro presenta il primo prototipo del sistema PAPRICA, evidenziandone pregi e difetti, illustra varie possibilità per la sua evoluzione e descrive la nuova versione, attualmente in progettazione; inoltre presenta anche alcune applicazioni in cui viene utilizzato con successo. Questo lavoro è corredato da alcune Appendici in lingua inglese che approfondiscono in dettaglio la descrizione dei problemi affrontati e delle soluzioni proposte.

Ringraziamenti

Desidero ringraziare in primo luogo Gianni Conte per avermi dato la possibilità di crescere intellettualmente, per avermi sempre incoraggiato nel mio lavoro e per avermi dimostrato ed insegnato che, volendo, la giornata lavorativa è formata da più di 24 ore.

Grazie anche a Giovanni Adorni, Vincenzo D'Andrea e Giulio Destri per avermi aiutato nel mio lavoro con utili discussioni, consentendomi di scambiare idee e di confrontarmi con diversi punti di vista e soprattutto per le serene giornate trascorse insieme.

Un grazie particolare va anche agli amici *torinesi* Francesco Gregoretti, Leonardo Reyneri e Claudio Sansoè per avere condiviso con me gioie e dolori del progetto PAPRICA e per l'amicizia che mi hanno dimostrato anche fuori dall'ambito universitario.

Last but not least, il mio più grande, fragoroso e incommensurabile ringraziamento lo rivolgo a Simona per avermi sopportato prima, durante e (spero) dopo la stesura di questo lavoro e per avermi sempre aiutato in ogni difficoltà.

Ai miei Genitori
A Simona

Indice

Indice delle Tabelle

Indice delle Figure

Capitolo 1

Introduzione

1.1 Motivazione

L'elaborazione delle immagini è una disciplina oggetto di ricerche ed applicazioni in molteplici e differenti campi che spaziano dalla navigazione automatica, alla robotica, alla medicina, alla prospezione geografica ed altro ancora.

I metodi utilizzati per elaborare automaticamente le immagini sono numerosi [8], in ogni caso però, è necessaria una fase di elaborazione di basso livello che, per definizione, non altera la struttura dati dell'immagine. Esempi di queste operazioni sono i primi filtraggi che vengono eseguiti sulle immagini subito dopo la loro acquisizione, il riconoscimento dei bordi delle superfici, lo smussamento dei livelli di grigio. Una caratteristica di gran parte delle operazioni di basso livello è il fatto che ogni elemento atomico (pixel) dell'immagine viene modificato secondo una regola fissa e locale, in maniera sincrona su tutti i pixel dell'immagine e dipendente solo dal proprio intorno (i pixel adiacenti).

Parallelamente, uno dei maggiori avanzamenti in campo informatico negli ultimi anni è stata la crescente disponibilità di calcolatori massivamente paralleli. Questa classe di calcolatori è caratterizzata dal parallelismo sui dati: essi sono costituiti da un numero molto elevato di processori elementari che eseguono contemporaneamente la medesima operazione, ognuno però elaborando dati diversi. Un esempio di calcolatore massivamente parallelo è la Connection Machine CM-2 [62], che può contenere da 2048 a 65536 processori elementari e che, per lo scambio dei dati tra i processori, possiede una rete di comunicazione configurabile a seconda delle esigenze del problema da risolvere. Nel caso dell'elaborazione di basso livello delle immagini il modello di comunicazione tra processori più conveniente è quello a griglia bidimensionale regolare, in quanto rispecchia la geometria dei pixel dell'immagine.

Il calcolo massivamente parallelo a griglia bidimensionale regolare può essere descritto convenientemente con il paradigma computazionale basato su *Morfologia Matematica* [88, 61] o su *Automi Cellulari* [97, 36, 46]. Poichè le architetture a parallelismo massivo con connessioni a griglia (*bidimensional mesh connected architectures*, MC^2) sintetizzano in hardware la topologia necessaria per ottimizzare le operazioni basate su tali paradigmi, il loro impiego produce un notevole incremento delle prestazioni.

Presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Parma, è in corso lo studio e la valutazione di PAPRICA [58, 37, 26, 23, 57] (PArallel PRocessor for Image Checking and Analysis), un'architettura SIMD (Single Instruction Muliple Data) a parallelismo massivo sviluppata in collaborazione con il Dipartimento di Elettronica del Politecnico di Torino. Un primo studio teorico ha consentito di fissare le specifiche del sistema; in seguito, con lo scopo di valutare eventuali modifiche prima della sua definitiva implementazione in hardware, è stato realizzato un simulatore del sistema completo su workstation tradizionale Unix e anche su Connection Machine CM-2, adatto quest'ultimo alla prototipazione di algoritmi grazie alla alta velocità di simulazione ottenibile da questa architettura. Successivamente è stato realizzato il primo prototipo hardware, che viene utilizzato ampiamente in diversi campi applicativi. Attualmente è in fase di verifica un secondo prototipo, le cui prestazioni sono state notevolmente migliorate grazie alla eliminazione di alcuni colli di bottiglia della precedente versione.

L'architettura PAPRICA è stata originariamente progettata per l'esecuzione di algoritmi di *Design Rule Checking* (DRC), per la verifica di layout di circuiti integrati e per l'elaborazione di immagini di tali circuiti [30]. La configurazione attuale di PAPRICA comprende 256 processori connessi a griglia bidimensionale, ognuno dei quali è collegato ai propri 8 processori adiacenti. Il set di istruzioni dei semplici processori single bit di PAPRICA è costituito sia da operatori puntuali che da operatori locali [8] e comprende anche alcune istruzioni frequentemente utilizzate nel campo del DRC. Mediante l'utilizzo degli operatori logici puntuali congiuntamente alle regole di traslazione di un'immagine nelle quattro direzioni principali, è facile dimostrare [22] che PAPRICA è in grado di sintetizzare qualunque Automa Cellulare definito su un vicinato comunque esteso.

1.2 Scelta Architetturale

La scelta di architetture a parallelismo massivo (SIMD) è stata motivata dalle seguenti considerazioni.

Il consumo di potenza dei sistemi dinamici può essere considerato proporzionale a CfV^2, dove C rappresenta le capacità del circuito, f è la frequenza del clock, e V è lo *swing* logico.

La potenza può essere ridotta agendo in tre modi differenti [48], minimizzando rispettivamente C, f, e V:

- utilizzando un maggiore livello di integrazione VLSI, e quindi riducendo la capacità C;
- sacrificando la capacità computazionale a vantaggio di un minore consumo di potenza, utilizzando frequenze di clock f ridotte (già implementato su numerosi personal computer portatili);
- riducendo la tensione di alimentazione V_{DD}.

Recentemente sono in fase di studio e valutazione nuove soluzioni tecnologiche per ridurre la tensione di alimentazione dei curcuiti integrati da 5 V a 3.3 V.

Sfortunatamente, però, questa riduzione porta come conseguenza anche la riduzione della velocità computazionale: per un *gate* CMOS [90], il ritardo T_d (assumendo un'approssimazione del primo ordine) è proporzionale a $\frac{V_{DD}}{(V_{DD} - V_T)^2}$; questo mostra che la riduzione di V_{DD} determina un incremento quasi lineare (fino al valore della soglia V_T) del ritardo del circuito T_d. Parallelamente, la riduzione di V_{DD} determina una riduzione quadratica del consumo di potenza. Di conseguenza, per ridurre il consumo di potenza è desiderabile operare alla frequenza più bassa possibile, ma per non degradare in maniera eccessiva le prestazioni di calcolo è necessario studiare un metodo di compensazione.

L'utilizzo di basse tensioni di alimentazione è discusso in [38], dove sono anche proposte diverse soluzioni architetturali per eliminare gli effetti indesiderati causati dalla riduzione di V_{DD}. La riduzione del consumo di potenza, senza modificare la potenza computazionale, può essere raggiunta mediante l'impiego di architetture SIMD a basso costo [35], composte da un numero elevato di processori elementari con frequenze di clock ridotte. L'impiego di basse frequenze di clock riduce il consumo di potenza a scapito di una maggiore occupazione d'area su silicio, mentre l'elevato numero di processori consente comunque di raggiungere potenze computazionali elevate.

Questa scelta architetturale è accompagnata dalle seguenti esigenze:

- l'approccio massivamente parallelo è basato sulla ripetizione su silicio dello stesso modulo elementare. In questo modo non si minimizza l'occupazione d'area su silicio, per cui tale soluzione non è indicata in problemi in cui le dimensioni fisiche del sistema devono essere contenute entro certi limiti.

- il parallelismo dell'architettura deve essere accompagnato da un maggior *routing* su chip, e quindi richiede una maggiore potenza; questa soluzione deve comunque essere considerata, valutata e ottimizzata attentamente;

- l'utilizzo di architetture massivamente parallele richiede il riprogetto degli algoritmi secondo un nuovo paradigma computazionale appropriato;

- poichè il numero di processori deve essere elevato, per contenere il più possibile le dimensioni del sistema, i processori elementari devono essere caratterizzati da una struttura estremamente semplice, e quindi possono eseguire solo operazioni semplici.

Capitolo 2

Breve Introduzione alla Morfologia Matematica

La Morfologia Matematica [71, 88, 61] è una disciplina che riguarda lo studio delle forme utilizzando la teoria insiemistica, e viene comunemente ed efficacemente utilizzata per la soluzione di problemi legati all'elaborazione delle immagini. La morfologia matematica viene generalmente impiegata per l'elaborazione di immagini a basso livello, poichè permette di filtrare e/o accentuare solo alcune caratteristiche delle immagini, in funzione della loro forma. A dimostrazione di quanto sia comunemente utilizzata, in letteratura si possono traovare numerosi trattati che ne dimostrano l'efficacia in diversi campi applicativi [61, 88, 112, 71, 104, 102, 108].

2.1 Concetti Principali

Nel contesto della morfologia matematica, un'immagine binaria A è definita come un sottoinsieme dello spazio Euclideo a due dimensioni E^2 $(Z \times Z)$:

$$A = \{a = (a_i, a_j) \mid a_i, a_j \in Z\} \tag{2.1}$$

Vengono definite tre operazioni unarie sulla generica immagine binaria A. Esse sono: il *complemento* $(\cdot)^c$, la *riflessione* $\breve{(\cdot)}$, e la *traslazione* $(\cdot)_t$: rispettivamente,

$$A^c \triangleq \{x \in E^2 \mid x \neq a, \text{ per ogni } a \in A\} \tag{2.2}$$

$$\breve{A} \triangleq \{x \in E^2 \mid x = -a, \text{ per qualche } a \in A\} \tag{2.3}$$

$$(A)_t \triangleq \{x \in E^2 \mid x = a + t, \text{ per qualche } a \in A\} \tag{2.4}$$

Inoltre, secondo le definizioni presentate in [61], alcuni dei più comuni operatori binari tra immagini sono: *espansione* $\oplus$, *erosione* $\ominus$, *apertura* $\circ$, *chiusura* $\bullet$, e *matching* $\oslash$: rispettivamente,

$$A \oplus B \triangleq \{x \in E^2 \mid x = a + b, \text{ per qualche } a \in A, b \in B\} \tag{2.5}$$

$$A \ominus B \triangleq \{x \in E^2 \mid x + b \in A, \text{ per ogni } b \in B\} \tag{2.6}$$

$$A \circ B \triangleq (A \ominus B) \oplus B \tag{2.7}$$

$$A \bullet B \triangleq (A \oplus B) \ominus B \tag{2.8}$$

$$A \oslash Q \triangleq (A \ominus Q_1) \cap (A^c \ominus Q_0) \ , \tag{2.9}$$

dove $B \in E^2$, e Q è una coppia $Q = (Q_0, Q_1)$, dove $Q_0, Q_1 \in E^2$, con il limite che $Q_0 \cap Q_1 = \emptyset$. Nelle espressioni precedenti A è l'immagine da elaborare, mentre B rappresenta l'*elemento strutturante*[1], cioè un altro sottoinsieme di E^2 la cui forma viene utilizzata per parametrizzare l'operazione.

Inoltre un elemento strutturante B si definisce *convesso* rispetto ad un insieme di altri elementi strutturanti $B_1, B_2, ..., B_n$, se e solo se esso può essere sintetizzato unicamente con operazioni di espansione tra gli elementi di $B_1, B_2, ..., B_n$:

$$B = \Big(\big((B_1 \oplus B_2) \oplus B_3\big) \oplus \ldots\Big) \oplus B_n \tag{2.10}$$

Negli ultimi anni sono state progettate e sviluppate un grosso numero di architetture dedicate all'elaborazione delle immagini mediante questo paradigma

[1]Nel caso delle operazioni di *matching*, l'elemento strutturante prende il nome di *matching template*.

computazionale. Una parte considerevole di esse sfrutta il parallelismo massivo secondo la classificazione SIMD, dove ogni pixel viene elaborato da un differente processore. Il problema principale di queste architetture è legato alla topologia di interconnessione tra i processori, che limita le dimensioni degli elementi strutturanti trattabili praticamente. In Appendice A sono considerate alcune diverse architetture (seriali, pipeline, e massivamente parallele) e per ognuna di esse è determinata la complessità computazionale richiesta dall'esecuzione di operazioni di base della morfologia matematica, quale l'espansione. Inoltre viene mostrato il problema della scomposizione ottima di un elemento strutturante non-convesso.

2.2 Morfologia Matematica e Automi Cellulari

L'unica sostanziale differenza tra l'approccio basato su Morfologia Matematica e quello basato su Automi Cellulari, è il seguente. Nel primo caso (*approccio insiemistico*) ogni cella del reticolo rappresenta un particolare elemento di un insieme e, di conseguenza, può assumere solo due configurazioni: presente ('1') o assente ('0'). A questo approccio binario si contrappone il secondo caso, in cui ogni cella può assumere un numero finito di configurazioni, generalmente maggiore di due.

Il numero di configurazioni ammesse per ogni cella contribuisce a definire l'insieme di operazioni che l'Automa può eseguire. Nel caso della Morfologia Matematica è costituito solo da operazioni binarie quali ad esempio l'unione o l'intersezione, mentre nel secondo caso più generale sono incluse anche funzioni più complesse il cui dominio di definizione è esteso a tutti gli stati ammessi dall'Automa.

E' chiaro però che, così come un processore single-bit (processore a parallelismo 1) può gestire una qualunque operazione su n bit, altrettanto facilmente, utilizzando algoritmi opportuni su un numero sufficiente di strutture dati (insiemi) bidimensionali, l'approccio basato su Morfologia Matematica può gestire una qualsiasi elaborazione su n bit.

Posto che entrambi gli approcci permettono l'esecuzione di qualunque operazione, la loro sostanziale differenza si riconduce all'efficienza di una loro eventuale implementazione in hardware. Tale efficienza dipende ovviamente dal tipo di applicazione richiesta. Nel caso dell'elaborazione di immagini a 256 toni di grigio (8 bit) mediante una macchina per Automi Cellulari in cui ogni cella può assumere 256 diverse configurazioni, l'efficienza dell'hardware è massima (100%), mentre scende al 12% nel caso di immagini binarie. Si consideri ad esempio il caso dell'elaborazione di immagini acquisite da autoveicolo, in cui sono richiesti diversi tipi di elaborazione che spaziano da filtraggi per l'eliminazione delle ombre (che operano su immagini a colori a 24 bit), a filtraggi per la riduzione del rumore (che operano invece su immagini a toni di grigio a 8 bit), fino ad arrivare ad elaborazioni prettamente morfologiche su immagini binarie.

Viceversa con un'architettura che implementi operazioni di Morfologia Matematica binaria l'efficienza hardware che si può raggiungere è sempre massima, qualunque sia il numero n di bit dell'immagine; per contro, il tempo di elaborazione aumenta di un fattore n per la manipolazione di immagini a n bit.

La scelta dell'architettura di elaborazione dipende dallo specifico campo di applicazione. Nel particolare caso esaminato (l'impiego di architetture massivamente parallele a bordo veicolo), gli obiettivi principali da conseguire sono la minimizzazione del costo e delle dimensioni fisiche del sistema e la massimizzazione della velocità di elaborazione. Il problema più vincolante è quello della minimizzazione del costo, e richiede, tra gli altri requisiti, la massimizzazione dell'efficienza hardware, ottenibile mediante l'utilizzo di un'architettura a singolo bit. Viceversa, la massimizzazione della velocità computazionale può essere facilmente conseguita con l'aumento modulare del numero delle unità di elaborazione integrate.

Capitolo 3

Il Sistema PAPRICA

Il progetto PAPRICA ebbe inizio nel 1988 con lo scopo di valutare l'utilizzo di architetture massivamente parallele [74, 49, 81, 62] e del paradigma computazionale basato su morfologia matematica per la risoluzione di problemi legati alla verifica di layout di circuiti integrati.

Il secondo passo è stato lo sviluppo di un primo prototipo di un'architettura formata da un insieme di processori elementari connessi tra loro mediante collegamenti a griglia bidimensionale. L'obiettivo principale era quello di avere un sistema potente, ma sufficientemente semplice da permettere una implementazione a basso costo e che potesse seguire l'evoluzione tecnologica. La prima versione del sistema hardware è disponibile dal 1992 ed è stata ampiamente utilizzata nel corso di questi anni.

Nel frattempo sono stati valutati anche altri campi applicativi dell'architettura PAPRICA, tra cui l'emulazione di reti neurali cellulari [52] e il suo utilizzo su autoveicoli per assistenza alla guida. Nell'ambito del progetto EUREKA PROMETHEUS nel 1991 è iniziato lo sviluppo di un veicolo di test nel campo automobilistico, ma i requisiti di velocità di elaborazione, dimensioni del sistema e consumo di potenza non erano compatibili con le prestazioni offerte dal primo prototipo.

Per questa ragione è stato sviluppato un secondo prototipo (PAPRICA-1/2) completamente compatibile a livello software con il precedente, ma le cui prestazioni sono aumentate di un fattore circa 5 e le cui dimensioni sono state ridotte di un fattore 4 rispetto alla precedente versione. Questa è la versione correntemente utilizzata dai laboratori dei due Dipartimenti coinvolti nel progetto (Dipartimento di Ingegneria dell'Informazione, Università di Parma e Dipartimento di Elettronica, Politecnico di Torino) e installata sul veicolo mobile MOB-LAB (vedi capitolo 5.1 e figura 5.1).

Al termine dell'implementazione del primo prototipo è stata avviata una valutazione critica del sistema per determinare i colli di bottiglia e l'influenza dei vari parametri sulle prestazioni complessive del sistema.

Tale revisione critica ha portato alla formulazione di un nuovo sistema, PAPRICA-3, in cui sono state modificate sia la struttura di interconnessione tra i processori, sia l'insieme delle istruzioni macchina. In particolare, la topologia è stata modificata da bidimensionale a monodimensionale, mentre l'insieme delle istruzioni è orientato verso un'implementazione RISC.

3.1 Il Paradigma Computazionale

Il paradigma computazionale che ha ispirato il progetto del sistema PAPRICA deriva dal lavoro di Serra [88], che ha introdotto il concetto di morfologia matematica come uno strumento teorico per la manipolazione di immagini memorizzate in *bit-map*. La morfologia matematica offre un approccio alla elaborazione di insiemi discreti di dati (immagini) basato sulla teoria insiemistica.

Il paradigma computazionale di PAPRICA è basato sul concetto di *matching operator* $\oslash$ introdotto nel campo della morfologia matematica, derivato dalla *hit-miss transform* descritta in [88]. Avendo definito la *immagine N-dimensionale* $\mathcal{F}_N \subset Z^N$ come un sottoinsieme convesso, limitato, e "rettangolare", dello spazio Euclideo discreto Z^N, un *matching element N-dimensionale semplice* è una coppia $\mathcal{Q} = (\mathcal{Q}_0, \mathcal{Q}_1)$, dove entrambi i $\mathcal{Q}_0, \mathcal{Q}_1 \subset \mathcal{F}_N$ con la limitazione che $\mathcal{Q}_0 \bigcap \mathcal{Q}_1 = \emptyset$. L'operazione di *matching elementare* con un matching element semplice è definito come:

$$A \oslash \mathcal{Q} \triangleq \{y \in \mathcal{F}_N \mid (y+b) \in A \quad \text{et} \quad (y+c) \notin A \ , \qquad \text{per ogni} \quad b \in \mathcal{Q}_1, c \in \mathcal{Q}_0\} \tag{3.1}$$

oppure, in termini dell'operatore di erosione:

$$A \oslash \mathcal{Q} \triangleq (A \ominus \mathcal{Q}_1) \bigcap (A^c \ominus \mathcal{Q}_0) \ . \tag{3.2}$$

Un *matching complementato* $\oslash^c$ è definito come:

$$A \oslash^c \mathcal{Q} \triangleq (A \oslash \mathcal{Q})^c = (A^c \oplus \check{\mathcal{Q}}_1) \bigcup (A \oplus \check{\mathcal{Q}}_0) \ . \tag{3.3}$$

Un *matching composito* con una *matching list* $\mathcal{Q}_{\mathcal{L}} = \{\mathcal{Q}^1, \ldots, \mathcal{Q}^k, \ldots, \}$ è l'unione di matching elementari:

$$A \oslash \mathcal{Q}_{\mathcal{L}} \triangleq \bigcup_i (A \oslash \mathcal{Q}^i) \ . \tag{3.4}$$

Un semplice matching element bidimensionale 3×3 $\mathcal{Q}$ può essere visualizzato utilizzando la seguente notazione: $\begin{array}{|c|c|c|}\hline x & x & x \\ \hline x & x & x \\ \hline x & x & x \\ \hline\end{array}$, dove "x" può essere 0, 1 o $-$ se il pixel corrispondente del matching element è un elemento di $\mathcal{Q}_0$, $\mathcal{Q}_1$ o di nessuno dei due. Il centro della matrice rappresenta il pixel $(0,0)$ di $\mathcal{Q}$ (cioè si assume implicitamente che l'origine delle due coordinate corrisponda al centro della matrice 3×3). Un *matching composito* può essere rappresentato da una lista di operatori semplici:

$$\mathcal{Q}_{\mathcal{L}} = \begin{array}{|c|c|c|}\hline x & x & x \\ \hline x & x & x \\ \hline x & x & x \\ \hline\end{array}, \begin{array}{|c|c|c|}\hline x & x & x \\ \hline x & x & x \\ \hline x & x & x \\ \hline\end{array}^{\,c}, \ldots, K\begin{array}{|c|c|c|}\hline x & x & x \\ \hline x & x & x \\ \hline x & x & x \\ \hline\end{array}, \ldots \tag{3.5}$$

dove $K \in [2, 4, 8]$. L'apice ^{c} dichiara che per quello specifico elemento un matching complementato deve essere eseguito al posto di $\oslash$, mentre la costante numerica K è una forma breve che indica la lista delle K possibili rotazioni dell'operatore elementare di $\frac{360}{K}$ gradi.

PAPRICA può eseguire delle operazioni di matching utilizzando un particolare insieme di elementi che forma l'*instruction set.* L'instruction set può sintetizzare qualunque matching element utilizzando semplici composizioni seguendo l'algebra della morfologia matematica.

Figura 3.1. Fotografia del cestello VME contenente le 2 schede del primo prototipo del sistema PAPRICA

3.2 L'Architettura Interna di PAPRICA

L'implementazione del primo prototipo dell'architettura (vedi figura 3.1) consiste di 2 schede VME: la prima contiene le memorie di programma e di immagine (4 MegaBytes di RAM statica) e la parte di interfaccia con il bus VME; l'altra contiene la matrice dei processori e l'unità di controllo ed è connessa alla prima come *piggy-back*. La seconda scheda (vedi figura 3.2) contiene 256 processori alloggiati in 4×4 chip contenenti ciascuno 4×4 processori. Tali chip sono stati progettati utilizzando una metodologia *full-custom* e realizzati con una tecnolo-

Figura 3.2. Fotografia della scheda VME contenente i 256 processori

gia CMOS da 1.5 μm, con una complessità totale di circa 35000 transistors su un'area di 45 mm^2. Sono alloggiati all'interno di PGA package con 85 pin, e la resa di produzione misurata è di circa il 65% su un insieme di 120 campioni. Il consumo medio di potenza è valutato di circa 0.3 W a 5 V.

La seconda versione è stata completamente reingegnerizzata e consiste di una sola scheda VME di dimensioni ridotte (6U) rispetto alle precedenti (9U); integra ancora 256 processori elementari e comprende 8 MegaBytes di RAM statica.

3.3 L'Architettura Esterna di PAPRICA

Come già accennato al principio del capitolo 3, il primo prototipo e la versione corrente sono completamente compatibili a livello software, e quindi condividono la stessa architettura esterna.

L'insieme delle istruzioni Assembly del sistema PAPRICA può essere suddiviso in tre classi:

- *Istruzioni di Esecuzione*: sono inviate dal controllore alla matrice dei processori; sono le istruzioni che servono per eseguire le operazioni morfologiche e logiche, e vengono eseguite contemporaneamente da ogni processore in un unico colpo di clock.

- *Istruzioni di Mapping*: sono intercettate dal controllore e servono per modificare in modo dinamico le aree di memoria immagine da cui il sistema di virtualizzazione legge e scrive l'immagine.

- *Istruzioni di Controllo del Flusso di Programma*: anche queste sono intercettate dal controllore e servono per modificare il flusso di programma, basandosi sullo stato di alcuni *flag* globali.

Un programma Assembly di PAPRICA è formato da un insieme di blocchi di UPDATE separati dalle istruzioni di `UPDATE`, che sono qui considerate appartenenti al terzo gruppo.

Istruzioni di Esecuzione

Una singola istruzione di esecuzione di PAPRICA è formata dalla concatenazione di due operatori ortogonali: un *graphic operator* e un *logical operator* e la sua sintassi è la seguente:

$$L_D \;=\; G(L_{S1})\ [*\ L_{S2}]\ [\%A]$$

Il primo (G) opera su un primo *layer sorgente* (L_{S1}) dell'immagine (utilizzando un vicinato 3×3), mentre il secondo calcola una funzione booleana tra il risultato dell'operatore grafico e il pixel centrale di un secondo *layer sorgente* (L_{S2}). Il risultato è memorizzato nel *layer destinazione* (L_D). %A è un parametro opzionale usato per accumulare (con una operazione di OR) il risultato dell'operazione sul layer L_0, utilizzato come accumulatore.

L'operatore grafico G è uno tra i sedici *match compositi* elencati nella tabella 3.1, mentre l'operatore logico '$*$' è uno degli otto operatori Booleani unari, binari, o ternari elencati nella tabella 3.2.

Istruzioni di Mapping

Il linguaggio Assembly di PAPRICA comprende alcune istruzioni che consentono di modificare i parametri richiesti per definire i trasferimenti di dati tra la memoria immagine e la matrice dei processori. L'elenco delle istruzioni di mapping è presentato nella tabella 3.3.

Istruzioni di Controllo del Flusso di Programma

Durante l'elaborazione, per ogni blocco di UPDATE vengono calcolati tre *flag* globali sulla base del risultato dell'ultima elaborazione. I tre *flag SET*, *RESET* e *NOCHANGE* vengono posti ad 1 rispettivamente se e solo se l'ultima istruzione del blocco di UPDATE ha generato tutti '1', '0' oppure non ha alterato lo stato del layer destinazione.

La tabella 3.4 mostra l'elenco di tutte le istruzioni per il controllo del flusso di programma, basate sullo stato dei 3 *flag* globali.

3.4 Il Meccanismo di Virtualizzazione

Due diversi meccanismi per la virtualizzazione dei processori sono generalmente implementati sulle achitetture a parallelismo massivo: la scelta dipende dalla

NOME	DESCRIZIONE	MATCHING TEMPLATE
NOP (L_{S_1})	No OPeration	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & 1 & - \\ \hline - & - & - \\ \hline\end{array}$
INV (L_{S_1})	INVersion	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & 0 & - \\ \hline - & - & - \\ \hline\end{array}$
NMOV (L_{S_1})	North MOVe	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & - & - \\ \hline - & 1 & - \\ \hline\end{array}$
SMOV (L_{S_1})	South MOVe	$\begin{array}{\|c\|c\|c\|}\hline - & 1 & - \\ \hline - & - & - \\ \hline - & - & - \\ \hline\end{array}$
WMOV (L_{S_1})	West MOVe	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & - & 1 \\ \hline - & - & - \\ \hline\end{array}$
EMOV (L_{S_1})	East MOVe	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline 1 & - & - \\ \hline - & - & - \\ \hline\end{array}$
EXP (L_{S_1})	EXPansion	$\begin{array}{\|c\|c\|c\|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array}^{c}$
VEXP (L_{S_1})	Vertical EXPansion	$\begin{array}{\|c\|c\|c\|}\hline - & 0 & - \\ \hline - & 0 & - \\ \hline - & 0 & - \\ \hline\end{array}^{c}$
HEXP (L_{S_1})	Horizontal EXPansion	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline 0 & 0 & 0 \\ \hline - & - & - \\ \hline\end{array}^{c}$
NEEXP (L_{S_1})	NorthEast EXPansion	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline 0 & 0 & - \\ \hline 0 & 0 & - \\ \hline\end{array}^{c}$
ERS (L_{S_1})	ERoSion	$\begin{array}{\|c\|c\|c\|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$
VERS (L_{S_1})	Vertical ERoSion	$\begin{array}{\|c\|c\|c\|}\hline - & 1 & - \\ \hline - & 1 & - \\ \hline - & 1 & - \\ \hline\end{array}$
HERS (L_{S_1})	Horizontal ERoSion	$\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline 1 & 1 & 1 \\ \hline - & - & - \\ \hline\end{array}$
NEERS (L_{S_1})	NorthEast ERoSion	$\begin{array}{\|c\|c\|c\|}\hline - & 1 & 1 \\ \hline - & 1 & 1 \\ \hline - & - & - \\ \hline\end{array}$
BOR (L_{S_1})	BORder	$8\begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & 1 & 0 \\ \hline - & - & - \\ \hline\end{array}$
LS2 (L_{S_1})	LesS than 2	$\left(4\begin{array}{\|c\|c\|c\|}\hline 1 & 1 & - \\ \hline 1 & 1 & - \\ \hline 1 & 1 & - \\ \hline\end{array}, 4\begin{array}{\|c\|c\|c\|}\hline 1 & 1 & - \\ \hline 1 & 1 & - \\ \hline - & - & 0 \\ \hline\end{array}, \begin{array}{\|c\|c\|c\|}\hline - & - & - \\ \hline - & 0 & - \\ \hline - & - & - \\ \hline\end{array}\right)^{c}$

Tabella 3.1. Elenco degli Operatori Grafici $G(L_{S_1})$ di PAPRICA

NOME	DESCRIZIONE	EQUIVALENTE A
(omesso)	Nessuna Operazione	$L_D = G(L_{S_1})$
$\sim$	NOT	$L_D = G(L_{S_1})^c$
&	AND	$L_D = G(L_{S_1}) \bigcap L_{S_2}$
&$\sim$	ANDNOT	$L_D = G(L_{S_1}) \bigcap (L_{S_2})^c$
$\mid$	OR	$L_D = G(L_{S_1}) \bigcup L_{S_2}$
$\mid \sim$	ORNOT	$L_D = G(L_{S_1}) \bigcup (L_{S_2})^c$
$\wedge$	EXOR	$L_D = (G(L_{S_1}) \bigcap (L_{S_2})^c) \bigcup \bigcup (G(L_{S_1})^c \bigcap L_{S_2})$
$+$	PLUS (Somma Aritmetica)	$L_D =$ "somma aritmetica" di $G(L_{S_1}) + L_{S_2} + L_0$ $L_0 =$ "riporto" della somma di $G(L_{S_1}) + L_{S_2} + L_0$

Tabella 3.2. Elenco degli Operatori Logici '$*$' di PAPRICA

quantità di memoria a disposizione di ogni processore.

Nel primo caso [62, 44, 45, 50, 51] (qui definito *Virtualizzazione Interna*) l'insieme completo dei dati viene memorizzato all'interno della memoria dei processori: l'insieme dei dati (immagine) viene suddiviso in sottoinsiemi non sovrapposti, e ogni sottoinsieme viene caricato nella memoria di un processore differente. L'elaborazione viene poi serializzata da ogni processore.

Viceversa, nel secondo caso [87, 74, 81, 37, 41, 42] (qui definito *Virtualizzazione Interna*) ogni processore ha una quantità di memoria che non gli permette di memorizzare un insieme di dati esteso come nel caso precedente. Di conseguenza l'insieme dei dati da elaborare viene partizionato in sottoinsiemi (sottofinestre) di dimensioni corrispondenti alla matrice dei processori; questi vengono caricati sequenzialmente nella memoria della matrice dei processori per essere elaborati. In questo caso l'elaborazione viene eseguita fino a che non si incontra una particolare istruzione (qui chiamata `UPDATE`). I risultati parziali ottenuti fino a questo punto vengono scaricati di nuovo nella memoria immagine. Questo processo viene ripetuto fino a che tutti i sottoinsiemi dell'immagine originale sono stati elaborati. Poi il primo sottoinsieme viene ricaricato nella matrice dei

ISTRUZIONE	SIGNIFICATO
MEMORG <*SizX*> <*SizY*> <*Par*>	Fissa le dimensioni della memoria immagine di PAPRICA.
SKIP.READ <*SkX*> <*SkY*> <*SkL*>	Fissa i parametri che servono per l'emulazione di architetture piramidali (per il sottocampionamento dell'immagine, ovvero il passaggio ai livelli superiori della piramide.
SKIP.WRITE <*SkX*> <*SkY*> <*SkL*>	Fissa i parametri che servono per l'emulazione di architetture piramidali (per il sovracampionamento dell'immagine, ovvero il passaggio ai livelli inferiori della piramide.
VALIDITY <*Area*>	Fissa l'area di validità per il successivo blocco di UPDATE.
LIMIT <*Area*>	Fissa l'area di lettura dalla memoria immagine per il successivo blocco di UPDATE.
MARGIN <*M*>	Definisce la dimensione lineare della zona della matrice dei procssori che è stata invalidata dall'esecuzione di operatori grafici.

Tabella 3.3. Elenco delle Istruzioni di Mapping di PAPRICA

processori e l'elaborazione viene proseguita fino alla nuova istruzione `UPDATE`.

I sistemi a basso costo generalmente utilizzano questa soluzione per la virtualizzazione dei processori, analizzata in dettaglio in [57, 29], che necessita di una memoria esterna per la memorizzazione dell'insieme dei dati. Uno dei problemi più grossi che si possono riscontrare nei sistemi con una memoria esterna, è dato dal consistente tempo di accesso ad una memoria esterna (fuori chip).

3.5 Vantaggi della Virtualizzazione Esterna

La seguente sezione presenta alcuni dei vantaggi offerti dalla virtualizzazione esterna e descrive le estensioni architetturali del sistema PAPRICA che consen-

ISTRUZIONE	SIGNIFICATO
UPDATE	Separa due blocchi di UPDATE. Tutte le altre istruzioni per il controllo del flusso di programma sono considerate delle istruzioni di UPDATE implicito.
FOR <*iter*> istruzioni ENDFOR	Ripete *istruzioni* <*iter*> volte.

IF [NOT] <*flag*>
istruzioni
 	
ELSE
istruzioni
ENDIF
	Esegue questo segmento di codice se [NOT] <*flag*> è vera.
	
	Altrimenti esegue questo segmento di codice.

REPEAT istruzioni UNTIL [NOT] <*flag*>	Ripete *istruzioni* fino a che [NOT] <*flag*> è vera.
CALL <*address*>	Chiama una procedura all'indirizzo <*address*>.
RET	Ritorna dalla procedura.
JUMP <*address*>	Salta alla locazione <*address*>.

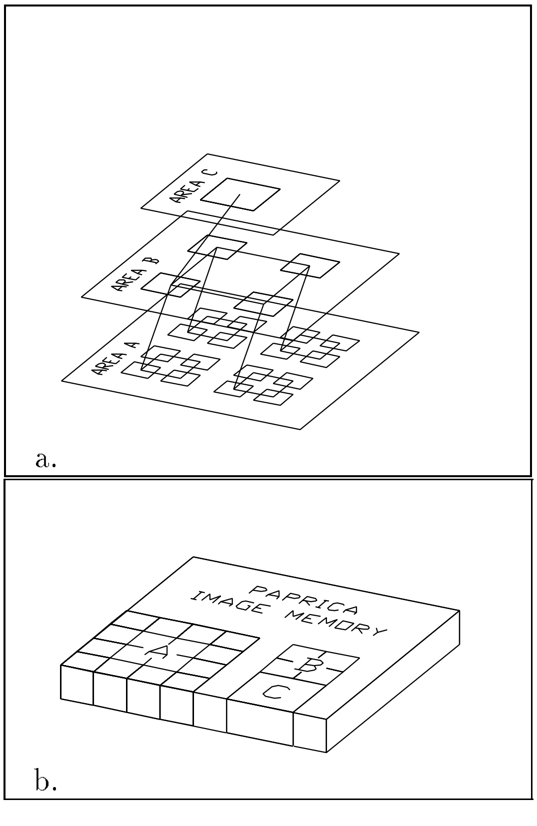
Tabella 3.4. Elenco delle Istruzioni di Controllo del Flusso di Programma di PAPRICA

tono di trarne beneficio.

Elaborazione Piramidale

Le architetture piramidali hanno mostrato grossi vantaggi nel campo dell'elaborazione delle immagini [32, 85, 92, 34, 93]. L'unico vero problema di questi tipi di architetture è dato dal fatto che non sono facilmente integrabili oltre una certa dimensione [6, 33]. Questo perchè le interconnessioni tra i processori hanno un'intrinseca struttura tridimensionale (vedi figura 3.3.a) che dà origine a problemi di integrazione quando devono essere realizzate sulla superficie bidimensionale del silicio.

PAPRICA risolve il problema delle interconnessioni grazie al fatto che l'hardware consente una definizione delle aree di lettura e di scrittura dalla memoria immagine modificabile dinamicamente (vedi il paragrafo 3.3). Utilizzando dei fattori di `SKIP` diversi da 1, è possibile emulare diverse architetture piramidali, come per esempio la topologia a "4 figli per genitore" mostrata in figura 3.3.a. La figura 3.3.b mostra la configurazione della memoria immagine associata a tale soluzione architetturale.




```
                                ; ==== 1st LAYER ====
LIMIT <AREA A>                  ; takes input data from area A
VALIDITY <AREA A>               ; writes results into area A
SKIP_READ 1 1                   ; one input and
SKIP_WRITE 1 1                  ; one output per PE
...
...                             ; 1st LAYER ROUTINE
...
                                ; ==== 2nd LAYER ====
LIMIT <AREA A>                  ; takes input data from area A
VALIDITY <AREA B>               ; writes results into area B
SKIP_READ N_Px N_Py             ; input data organized
                                ; as N_Px N_Py pixels,
                                ; where N_Px N_Py = N_P
SKIP_WRITE 1 1                  ; just one output per each PE
                                ; (can be more than one)
...
...                             ; 2nd LAYER ROUTINE
...
                                ; ==== 3rd LAYER ====
LIMIT <AREA B>                  ; takes input data from area B
VALIDITY <AREA C>               ; writes results into area C
SKIP_READ N_Px N_Py             ; input data organized
                                ; as N_Px N_Py pixels,
                                ; where N_Px N_Py = N_P
SKIP_WRITE 1 1                  ; just one output per PE
                                ; (can be more than one)
...
...                             ; 3rd LAYER ROUTINE
...
                                ; == OTHER LAYERS ==
LIMIT ...                       ; same as above
...                             ;
```

c.

Figura 3.3. Simulazione di architettura piramidale con PAPRICA

Il livello inferiore della piramide (cioè l'insieme delle foglie dell'albero) è composto da C_P^{L-1} processori figli (dove L e C_P sono rispettivamente il numero di livelli e il numero di figli per genitore). Ogni processore della piramide ha accesso ad un totale di $N_{Px} \times N_{Py}$ pixel dell'immagine, che sono raggruppati in modo adiacente in un'area rettangolare (A) della memoria immagine di PAPRICA. I C_P^{L-2} processori del penultimo livello sono associati ad una regione rettangolare più piccola nella memoria immagine di PAPRICA (B). I processori di tutti i livelli sovrastanti, fino al processore alla radice, sono associati ad aree sempre più piccole (C, etc. etc.).

È chiaro che la sola limitazione alla dimensione della piramide è data dalla quantità di memoria immagine disponibile, mentre non c'è limite alla complessità delle interconnessioni tra i processori della piramide, che possono essere indifferentemente connessi ai fratelli, figli, o genitori.

La figura 3.3.c mostra il codice Assembly di PAPRICA necessario per emulare una struttura piramidale. Al principio il programma elabora il livello più basso della piramide, eseguendo filtraggi iniziali dei dati in ingresso. Questo è possibile avendo le aree di LIMIT e VALIDITY coincidenti con i dati in ingresso. Poi, `VALIDITY` e `SKIP` vengono modificati in modo da puntare all'area di memoria B e per ottenere la quantità di sottocampionamento necessaria (cioè 2×2), così che i dati di uscita dell'ultimo livello sono utilizzati come dati di ingresso per l'elaborazione del penultimo livello. Questo procedimento viene iterato, cambiando ogni volta `LIMIT`, `VALIDITY`, e `SKIP`, fino ad arrivare al livello della radice. È comunque da notare che l'elaborazione dei diversi livelli può essere anche completamente differente e che, cambiando solo i parametri di mapping, è possibile variare la topologia delle interconnessioni logiche. In ogni caso la capacità computazionale richiesta per l'emulazione di architetture piramidali è piccola in quanto i dati non devono essere mossi in memoria, ma viene modificata solo la modalità di accesso alla memoria.

Area di Attenzione

Talvolta, dopo aver caricato una sottofinestra dell'immagine nella matrice dei processori, l'elaborazione può essere inutile, e quindi l'operazione di salvataggio nella memoria immagine dei dati elaborati non è più richiesta. Questo fatto permette di diminuire la complessità computazionale, eliminando l'elaborazione delle finestre che non producono risultati significativi, e implementando quindi una sorta di "*focus of attention*" multiplo [75, 109].

La scelta di elaborare o meno una sottofinestra dipende dalle caratteristiche dell'insieme completo dei dati che formano la sottofinestra stessa. Per l'implementazione di questa estensione è stata proposta una modifica al controllore di PAPRICA, basata sulla considerazione che ogni pixel dell'immagine transita attraverso il bus dei dati (16 bit) che connette la memoria immagine e la matrice dei processori, e può essere catturato da un circuito addizionale. Questo circuito, sulla base dei dati che formano la sottofinestra caricata nella matrice dei processori, decide se eseguire l'elaborazione o meno.

L'elaborazione ottenuta in questo caso è sostanzialmente piramidale: un singolo valore viene calcolato per ognuna delle sottofinestre in cui viene suddivisa l'immagine; l'insieme di questi valori forma una maschera a risoluzione ridotta. Il valore associato ad ogni sottofinestra seleziona la particolare elaborazione (o addirittura nessuna elaborazione) che verrà eseguita dalla matrice dei processori.

Questa estensione al controllore di PAPRICA può essere utile sia per aumentare la velocità di elaborazione, considerando solo le aree di interesse, sia per differenziare l'elaborazione in zone diverse dell'immagine [29]. Un'analisi più dettagliata del problema è riportata nell'Appendice B.

3.6 Svantaggi della Virtualizzazione Esterna

Organizzazione della Parola nella Memoria

La scelta dell'organizzazione logica della memoria esterna (memoria immagine) è un parametro critico nel progetto di una architettura come PAPRICA. Due diverse soluzioni sono possibili, come mostrato in figura 3.4. Supponendo che l'immagine bidimensionale sia sul piano (x, y), nel primo caso (figura 3.4.a) la

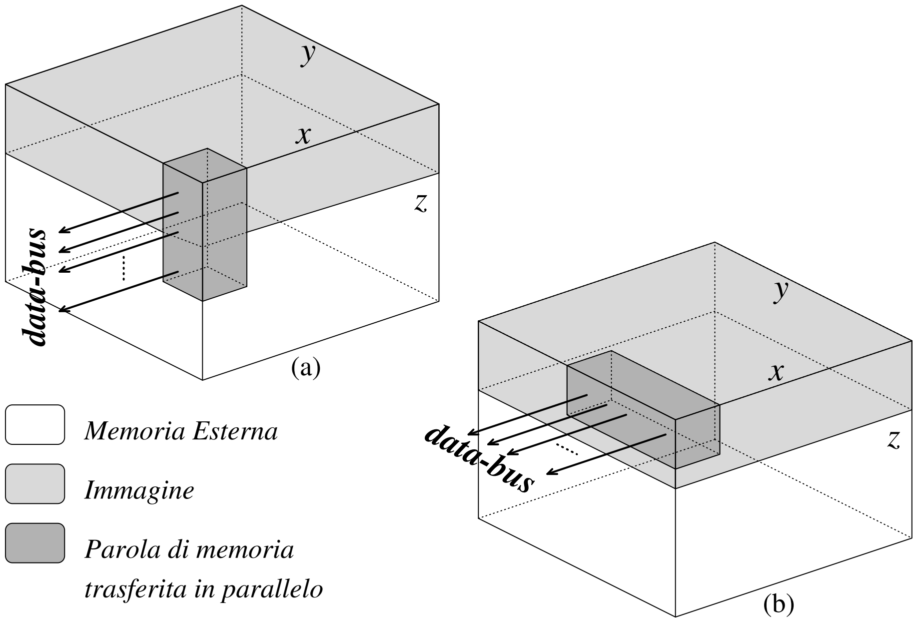


Figura 3.4. Le due diverse soluzioni per l'organizzazione logica della memoria esterna

parola di memoria è posta nella direzione z, ortogonale al piano dell'immagine. Nel secondo caso, invece, la parola di memoria è posta in una delle direzioni x o y.

Il primo caso ha lo svantaggio che per ogni trasferimento, viene mosso tra la memoria immagine e la matrice dei processori un numero fisso di bit (corrispondente al parallelismo del bus p_{BUS}), indifferentemente dalle richieste dell'applicazione. Come conseguenza, è poco probabile che l'efficienza del bus raggiunga valori elevati, in quanto generalmente il parallelismo della struttura dati è diverso da quello del bus. Viceversa, nel secondo caso (figure 3.4.b) vengono trasferite solo le informazioni strettamente necessarie e di conseguenza l'efficienza del bus raggiunge spesso valori elevati. Questa soluzione, però, richiede una estensione architetturale: infatti, nel caso dell'elaborazione delle immagini, ogni elemento dell'immagine (multi-bit pixel) viene reso disponibile dal sistema di acquisizione (telecamera, VCR, scanner,...) in parallelo, secondo lo schema di figura 3.4.a. Ogni elemento, infatti, porta l'informazione di un singolo pixel, e la seconda soluzione richiederebbe un meccanismo hardware per la trasposizione dei dati (*data shuffler*), come avviene ad esempio nella Connection Machine CM-2 [62] (*Sprint-chip*).

La prima versione del sistema PAPRICA utilizza la soluzione mostrata nella figura 3.4.a, ma nella versione successiva [21] (vedi capitolo 4.3) la memoria esterna è organizzata come in figura 3.4.b.

L'Appendice C presenta l'analisi dell'efficienza del bus dei dati in sistemi che utilizzano la prima soluzione (figura 3.4.a). Il risultato principale di questa analisi è l'individuazione di una tecnica (formalizzata in una procedura automatica per PAPRICA) finalizzata all'incremento dell'efficienza dell'hardware in modo da ottenere una maggiore velocità computazionale.

L'algoritmo proposto in Appendice C:

(a) riorganizza l'immagine contenuta nella memoria immagine;

(b) esegue l'elaborazione sull'immagine riorganizzata, in modo da sfruttare il più efficientemente possibile le caratteristiche della macchina fisica; e

(c) risistema il risultato nella memoria immagine nel formato originale;

Elaborazione ai Bordi della Matrice dei Processori

Il problema principale del meccanismo di virtualizzazione esterna è legato all'impossibilità per i processori del bordo della matrice di accedere ai propri vicini. Infatti, dopo l'esecuzione di una operazione morfologica che richiede l'accesso ad un vicinato 3×3, i valori memorizzati nei processori del bordo della matrice di elaborazione contengono dati non significativi. Quindi, solo una porzione della matrice dei processori (qui definita *Area di Validità*) contiene dati significativi. Di conseguenza la sottofinestra successiva che verrà trasferita nella matrice di elaborazione sarà parzialmente sovrapposta alla precedente in modo da determinare i valori che nell'elaborazione precedente non era stato possibile valutare.

3.7 Analisi delle Prestazioni

Quando i dati da elaborare sono immagini, la velocità computazionale S_{pr} è definita come il numero di pixel elaborati rapportato al tempo di elaborazione totale per una determinata sequenza di operazioni:

$$S_{pr} \triangleq \frac{\text{Numero\_di\_pixels\_elaborati}}{\text{Tempo\_di\_elaborazione}} \ . \tag{3.6}$$

Il caso ideale:

La massima velocità di elaborazione si può raggiungere ovviamente quando la matrice dei processori è in grado di contenere l'immagine completa, o quando la sequenza di istruzioni non riduce l'area di validità. In questo caso ideale, il numero di pixel elaborati coincide con il numero di pixel che formano l'immagine ($F \times F$), mentre il *tempo di elaborazione* t è dato dalla somma di tre contributi:

(a) il *tempo di caricamento* dell'immagine dalla memoria alla matrice dei processori;

(b) il *tempo di esecuzione* del programma;

(c) il *tempo di salvataggio* dei rislultati in memoria immagine.

I tempi di *caricamento* e *salvataggio* possono essere considerati uguali e dati dal numero di trasferimenti elementari (Q^2, dove Q rappresenta la dimensione lineare della matrice dei processori) moltiplicato per il tempo richiesto da un singolo trasferimento (T_M).

Il *tempo di esecuzione* è dato dal numero di istruzioni (L) moltiplicato per la durata di una singola istruzione (T_C) e quindi:

$$t = 2\ Q^2\ T_M + L\ T_C \ . \tag{3.7}$$

La velocità di elaborazione è quindi data da:

$$S_{pr} = \frac{Q^2}{2\ Q^2\ T_M + L\ T_C} \ . \tag{3.8}$$

Supponendo che T_C e T_M siano dello stesso ordine di grandezza, se L è trascurabile rispetto a $2\ Q^2$, la precedente relazione si riduce a:

$$S_{pr} \simeq \frac{Q^2}{2\ Q^2\ T_M} = \frac{1}{2\ T_M}\ . \tag{3.9}$$

L'equazione (3.9) mostra che quando il *tempo di esecuzione* è trascurabile (cioè nel caso ideale), allora il *tempo di elaborazione* complessivo è dato dal *tempo di trasferimento*: un problema la cui versione sequenziale era "CPU-bounded" è ora ridotto ad un problema "I/O-bounded", grazie all'uso di questo tipo di architettura massivamente parallela.

Il caso reale:

Si consideri ora il caso in cui il numero di processori è minore del numero di pixel dell'immaigne e si supponga che il programma contenga alcune istruzioni che riducono l'area di validità. Il parametro G, associato ad ogni istruzione, misura la riduzione dell'area di validità che si ottiene dopo la sua esecuzione: la figura 3.5 mostra alcune istruzioni e il loro corrispondente fattore G.

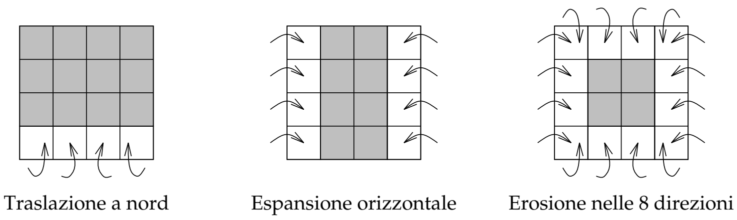


Figura 3.5. Il fattore G e, in grigio, l'area di validità dopo l'esecuzione della specifica istruzione

Lo scopo dell'istruzione `UPDATE` è di evitare che l'area di validità venga ridotta a zero. È ovvio che quando la frequenza delle operazioni `UPDATE` aumenta, allora aumenta anche il tempo di elaborazione a causa del tempo richiesto per il trasferimento della stessa finestra più volte tra la memoria e la matrice dei processori. Viceversa, anche se il numero di operazioni `UPDATE` è molto ridotto il tempo complessivo di elaborazione è elevato a causa della forte riduzione dell'area di validità e quindi del processamento di dati non significativi. Di conseguenza esiste un valore ottimo del numero di operazioni di `UPDATE` all'interno di un programma che massimizza la velocità di elaborazione.

Supponendo che le dimensioni dell'immagine da elaborare ($F \times F$) siano maggiori delle dimensioni della matrice dei processori ($Q \times Q$), allora il *tempo di esecuzione* può essere calcolato come il prodotto tra il *tempo di esecuzione* per una *singola* finestra (t_{1w}) e il numero di finestre da elaborare (n_w): $t = n_w\ t_{1w}$.

Questa volta il numero di finestre non è più semplicemente $\frac{F^2}{Q^2}$, perchè, a causa della riduzione dell'area di validità, le finestre devono essere sovrapposte. n_w è infatti dato dalla seguente espressione:

$$n_w = \frac{F^2}{Q'^2}\ , \tag{3.10}$$

dove la *dimensione effettiva* delle sottofinestre (Q') è data da:

$$Q' = Q - \frac{2\ G}{n_{upd}}\ . \tag{3.11}$$

Il rapporto $\frac{2\ G}{n_{upd}}$ mostra la riduzione lineare media dell'area di validità data da un programma formato da un insieme di istruzioni con un fattore di riduzione complessivo G, in cui sono state inserite n_{upd} istruzioni `UPDATE`.

Alcuni casi particolari mostrano la plausibilità dell'espressione precedente:

- Se $G = 0$, Q' si riduce a Q (figura 3.6.a);
- Se $G > 0$ e $n_{upd} = 1$ (una istruzione `UPDATE` al termine del programma), $Q' = Q - 2\ G$ (figura 3.6.b);
- Se $G > 0$ e $n_{upd} = 2$, (due istruzioni `UPDATE`), $Q' = Q - 2\ \frac{G}{2}$ (figura 3.6.c).

Il tempo totale di elaborazione di ogni singola sottofinestra (t_{1w}) è dato dal numero di blocchi di UPDATE moltiplicato per il tempo di elaborazione tra due `UPDATE` successivi.

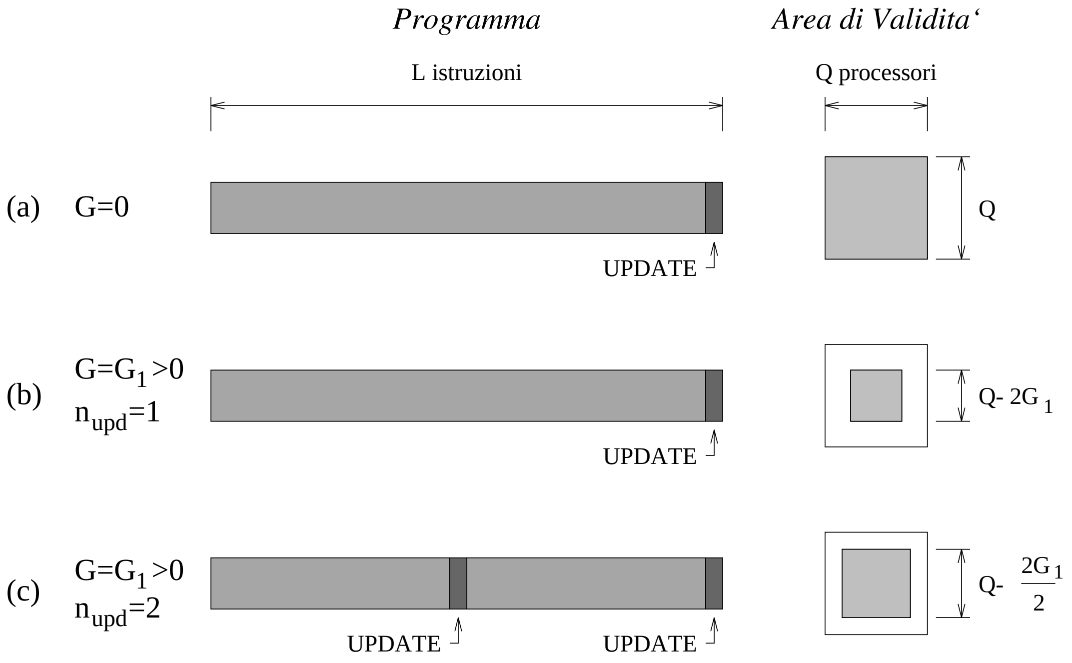


Figura 3.6. Riduzione dell'area di validità al termine di ogni blocco di UPDATE

Questo tempo di elaborazione è dato dalla somma del tempo di lettura e scrittura dei dati, più il tempo di esecuzione delle $\frac{L}{n_{upd}}$ istruzioni che compongono un blocco di UPDATE medio. Quindi il tempo di elaborazione è dato da:

$$t = n_w \; t_{1w} \; , \tag{3.12}$$

che diventa:

$$t = \left[\frac{F^2}{\left(Q - \frac{2\,G}{n_{upd}} \right)^2} \right] \times \left[n_{upd} \left(2\; Q^2\; T_M + \frac{L}{n_{upd}}\; T_C \right) \right] . \tag{3.13}$$

Dopo alcuni passaggi algebrici, l'equazione (3.6) diventa

$$S_{pr} = \frac{\left(Q - \frac{2\,G}{n_{upd}} \right)^2}{2\; Q^2\; T_M\; n_{upd} + L\; T_C} \; . \tag{3.14}$$

Si noti che l'equazione (3.8) può anche essere ottenuta dalla precedente come caso particolare quando $n_{upd} = 1$ (una singola operazione di `UPDATE` al termine del programma) e $G = 0$ (come nel caso ideale).

La velocità di elaborazione è quindi funzione di parametri che dipendono:

1. dalla tecnologia e dall'architettura del sistema (T_M, T_C, Q),
2. dalla particolare applicazione (L, G), e
3. dal numero (n_{upd}) (e dalla posizione) delle operazioni di `UPDATE`.

Fintanto che $L\ T_C$ è trascurabile rispetto a $2\ Q^2\ T_M\ n_{upd}$ - per esempio quando Q è sufficientemente grande - la velocità di elaborazione non dipende dalla lunghezza del programma. Inoltre, poichè i parametri di sistema (Q, T_M, T_C) sono fissati dalla architettura fisica, e il fattore G è determinato dall'applicazione specifica, la velocità di elaborazione può essere modificata agendo sul numero (e posizione) delle operazioni di `UPDATE`.

Miglioramento delle Prestazioni

Come già mostrato precedentemente, quando la matrice dei processori è sufficientemente grande la velocità di elaborazione dipende in maniera lieve dal numero di istruzioni L che formano il programma. Viceversa, la velocità di elaborazione dipende molto pesantemente dal numero e dalla posizione delle istruzioni `UPDATE`. Il prossimo paragrafo mostra un metodo per la determinazione del numero e della posizione ottima delle istruzioni `UPDATE`.

Posizionamento ottimo delle istruzioni `UPDATE`

Per ottenere il valore di n_{upd} che massimizza S_{pr} nella equazione (3.14), si consideri per semplicità il caso in cui $2\ Q^2\ T_M\ n_{upd} \gg L\ T_C$. In questo caso l'equazione (3.14) può essere approssimata rimuovendo il termine $L\ T_C$. Quindi, i massimi di S_{pr} possono essere determinati come le soluzioni della seguente derivata:

$$\frac{\partial}{\partial\ n_{upd}}\ S_{pr} \simeq \frac{\partial}{\partial\ n_{upd}}\ \frac{\left(Q - 2\ \frac{G}{n_{upd}}\right)^2}{2\ Q^2\ T_M\ n_{upd}} = 0\ . \tag{3.15}$$

Si definisce ora il parametro x come il rapporto tra la dimensione lineare della matrice dei processori (Q) e la riduzione media dell'area di validità all'interno di ogni blocco di UPDATE $\left(2\ \frac{G}{n_{upd}}\right)$:

$$x \triangleq \frac{Q\ n_{upd}}{2\ G}\ . \tag{3.16}$$

L'intervallo di variabilità di x è dato da: $1 < x \leq \frac{Q\ L}{2\ G}$, dove i valori limite si riferiscono a:

- $x = 1 \quad \Longrightarrow \quad Q = 2\ \frac{G}{n_{upd}}$:
  in questo caso l'area di validità svanisce completamente e S_{pr} si riduce a 0.
- $x = \frac{Q\ L}{2\ G} \quad \Longrightarrow \quad n_{upd} = L$:
  in questo caso ogni istruzione del programma è seguita da una istruzione `UPDATE`. $n_{upd} > L$ porterebbe ad avere un programma con due o più istruzioni `UPDATE` adiacenti, palesemente insensato.

Utilizzando il parametro x, l'equazione (3.15) diventa

$$\frac{\partial}{\partial\ x}\ \left[\frac{1}{8\ T_M}\ \frac{Q^2}{G^2}\ \frac{(x-1)^2}{x^3}\right] = 0\ , \tag{3.17}$$

e le sue due soluzioni sono:

$$\begin{cases} x_1 = 1 \quad \Longrightarrow \quad \left(n_{upd}\right)_1 = 2\ \frac{G}{Q} \\ \\ x_2 = 3 \quad \Longrightarrow \quad \left(n_{upd}\right)_2 = 6\ \frac{G}{Q} \end{cases} \tag{3.18}$$

Il grafico in figura 3.7 (S_{pr} vs x) mostra che la prima soluzione corrisponde ad un minimo, mentre la seconda identifica un massimo. Il valore ottimo per n_{upd} è quindi dato da:

$$\left[n_{upd}\right]_{opt} = 6\ \frac{G}{Q}\ . \tag{3.19}$$

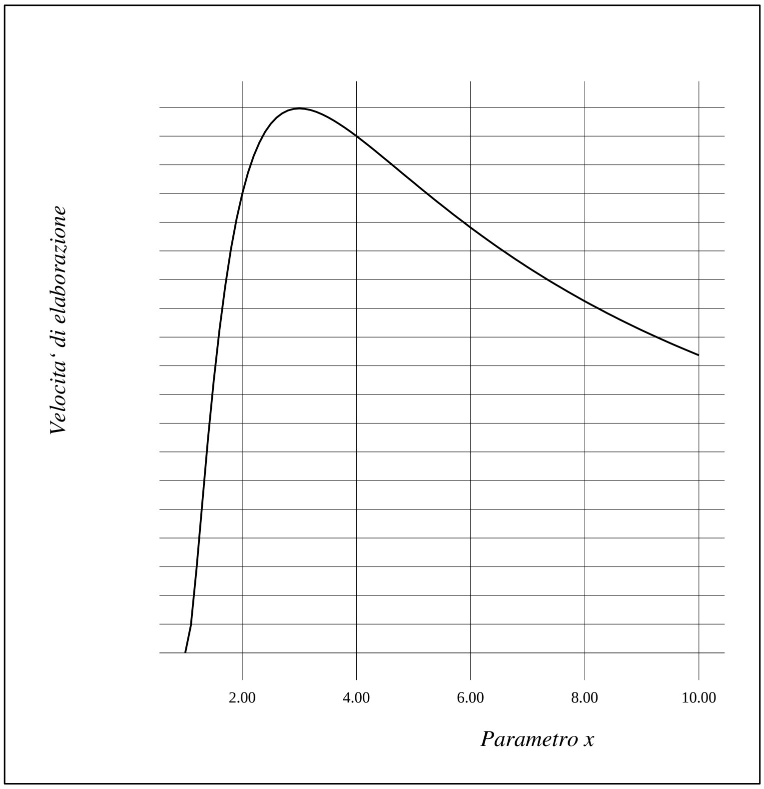


Figura 3.7. La velocità di elaborazione in funzione del parametro x

La dimensione dell'area di validità (Q') nel caso di posizionamento ottimo delle istruzioni `UPDATE` diventa:

$$Q' = \frac{2}{3} Q , \tag{3.20}$$

che corrisponde ad avere i $\frac{4}{9}$ dei processori che contengono risultati validi.

Per semplificare l'attività del programmatore, è stato sviluppato un algoritmo [14] per inserire automaticamente le istruzioni `UPDATE` in un programma Assembly per PAPRICA, con lo scopo di ottimizzare la velocità di esecuzione. L'algoritmo è semplicemente basato sulla scansione del programma in modo da contare il numero di processori che contengono risultati non validi, come funzione del numero di istruzioni eseguite. Una istruzione `UPDATE` viene inserita automaticamente quando i $\frac{5}{9}$ dei processori contengono risultati parziali non validi.

Un vantaggio ulteriore dell'algoritmo è la completa trasparenza del meccanismo di virtualizzazione nei confronti del programmatore, che non deve preoccuparsi di gestire la virtualizzazione dei processori.

Lo stesso risultato può essere raggiunto considerando l'efficienza della matrice dei processori η_{PA} nell'elaborazione di una singola sottofinestra $(Q \times Q)$ dell'immagine. L'efficienza della matrice dei processori η_{PA} è definita come il rapporto tra il numero di processori che producono risultati validi all'interno di un singolo blocco di UPDATE e il numero totale di processori. Nel caso reale $(G > 0)$, il numero di risultati validi è dato da $\left(Q - 2\,\frac{G}{n_{upd}}\right)^2$ e viene utilizzato per determinare l'efficienza della matrice dei processori nell'eseguire le $\frac{L}{n_{upd}}$ istruzioni che formano un blocco di UPDATE medio, su un insieme di $Q \times Q$ dati:

$$\eta_{\mathrm{PA}} = \frac{\left(Q - 2\,\frac{G}{n_{upd}}\right)^2}{Q^2}\ . \tag{3.21}$$

L'efficienza η_{TOT} dell'elaborazione complessiva (formata da n_{upd} blocchi di UPDATE) viene definita come il rapporto tra il numero di risultati validi generati dall'insieme completo dei blocchi di UPDATE $\left(Q - 2\,\frac{G}{n_{upd}}\right)^2$ e il numero di dati utilizzati per produrre il risultato $\left(n_{upd}\ Q^2\right)$:

$$\eta_{\mathrm{TOT}} \triangleq \frac{\left(Q - 2\,\frac{G}{n_{upd}}\right)^2}{Q^2\ n_{upd}} = \frac{\eta_{\mathrm{PA}}}{n_{upd}}\ . \tag{3.22}$$

Questa espressione è equivalente all'equazione (3.14), tenuto conto dell'approssimazione fatta al termine del paragrafo 3.7. Questo mostra che la massimizzazione dell'efficienza del sistema η_{TOT} corrisponde alla massimizzazione della velocità di elaborazione S_{pr}.

3.8 L'Abingdon Cross Benchmark

Lo scopo principale dell'*Abingdon Cross benchmark* [80] è quello di comparare le prestazioni di architetture di sistemi dedicati all'elaborazione di immagini. Questo benchmark è stato definito per essere utilizzato su ogni tipo di architettura, indifferentemente dal suo paradigma computazionale. L'obiettivo del benchmark è il calcolo del tempo necessario per il filtraggio di un'immagine di test (8 bit/pixel); il filtraggio consiste nella determinazione dello scheletro [8] di una croce sintetica a cui è stato aggiunto del rumore bianco gaussiano a media nulla (vedi figura D.1.a). Il rapporto segnale-rumore è di 0 dB.

Non ci sono limitazioni sulla particolare implementazione dell'Abingdon Cross benchmark, ma le migliori prestazioni vengono ovviamente raggiunte quando l'algoritmo scelto corrisponde al paradigma computazionale della macchina. Inoltre il progetto dell'algoritmo può tenere conto della conoscenza dei seguenti parametri utilizzati nella generazione dell'immagine sintetica di test.

1. Numero di toni di grigio: 256 (8 bit/pixel);
2. Dimensione lineare dell'immagine: N;
3. Valor medio del rumore Gaussiano aggiunto: 0;
4. Rapporto segnale rumore: 0 dB.
5. Dimensione della croce relativa alla dimensione dell'immagine: lunghezza del braccio: $\frac{3}{4}N$; larghezza del braccio: $\frac{N}{8}$;

Un esempio dell'immagine di test con $N = 128$ è presentato in figura D.1.a. Il fattore di qualità QF (cioè l'indice delle prestazioni di un particolare sistema) è definito come il rapporto tra la dimensione dell'immagine di test (N) e il tempo richiesto per filtrarla (T):

$$QF = \frac{N}{T} \; . \tag{3.23}$$

Per sistemi commerciali viene definito anche il rapporto prezzo-prestazioni, ma poichè PAPRICA non è un sistema commerciale, ci si limita qui a determinare solo il fattore di qualità. L'algoritmo completo è presentato nell'Appendice D insieme ai risultati parziali dell'elaborazione, mentre in tabella 3.5 è riportato da [80] il fattore di qualità di alcuni sistemi non commerciali. In tabella è incluso anche il fattore di qualità del primo prototipo di PAPRICA e della seconda versione (PAPRICA-1/2).

Sistema	Fattore di qualità
CLIP4 (96×96)	$7.3\ 10^3$
Cyto III	$1.0\ 10^2$
DIP	4.7 10
PHP	7.1
POP II	2.9
VAP	1.8 10
WARP	$5.0\ 10^2$
PAPRICA-1/1	$6.4\ 10^2$
PAPRICA-1/2	$1.1\ 10^3$

Tabella 3.5. Il fattore di qualità di alcuni sistemi non commerciali

Capitolo 4

L'Evoluzione del Sistema PAPRICA

A partire dal primo prototipo, il sistema PAPRICA è stato analizzato a fondo sia dal punto di vista teorico sia dal punto di vista pratico per determinare eventuali modifiche allo scopo di eliminare i problemi descritti nel paragrafo 3.6. In sostanza, gli studi di fattibilità hanno mostrato tre diverse soluzioni possibili, che verranno descritte brevemente in questo capitolo.

- La prima consiste nel mantenere la stessa architettura del prototipo originale e, sfruttando l'evoluzione tecnologica e l'eventuale ottimizzazione del progetto dei chip, nell'incrementare la dimensione della matrice dei processori. Se la dimensione potesse essere aumentata fino a diventare dello stesso ordine di grandezza delle immagini da elaborare, la riduzione delle prestazioni causata dal caricamento e scaricamento dei dati tra la memoria immagine e la matrice dei processori verrebbe limitata.

- La seconda soluzione è basata sull'incremento dello spazio di indirizzamento di ogni singolo processore per mezzo di elementi di memoria esterni. Questa scelta semplifica il meccanismo di virtualizzazione ma limita la dimensione della matrice dei processori e presenta alcuni problemi ulteriori legati all'esecuzione di operazioni morfologiche.

- Infine, la terza soluzione è basata sulla modifica della struttura interna della matrice dei processori: da bidimensionale a monodimensionale. La virtualizzazione è ora eseguita scandendo le diverse righe dell'immagine, e non più suddividendo l'insieme dei dati in finestre.

4.1 Estensione del Progetto Originale

Il progetto dei chip del primo prototipo del sistema PAPRICA è stato eseguito in modo estremamente conservativo, in termini di tecnologia, layout, e numero di processori per chip. Il chip originale contenente 4×4 processori occupa un'area di 6.5 mm $\times$ 6 mm e utilizza una tecnologia CMOS a 1.5 μm, mentre un recente riprogetto dello stesso con una tecnologia leggermente migliorata ha portato ad un'area di approssimativamente 5 mm $\times$ 5 mm. Quindi, con una tecnologia CMOS a 1.5 μm è possibile integrare una matrice di 8×8 processori in un singolo chip di area approssimativamente 10 mm $\times$ 10 mm con una resa accettabile.

Con una tecnologia PCB standard un array di 4×4 chip porterebbe ad una scheda con 32×32 processori, ancora troppo piccola per eliminare le limitazioni delle prestazioni.

Uno studio di fattibilità ha mostrato che con tecnologie da 1 e 0.8 μm sarebbe possibile integrare una matrice di 16×16 processori su un unico *dice*. In questo caso i problemi sarebbero solo spostati al livello più basso, in quanto ogni chip avrebbe un numero di pin elevatissimo (più di 200) e sia la potenza di alimentazione sia la velocità comincerebbero a dipendere in maniera forte dalle capacità a livello di scheda.

Inoltre sono state analizzate anche delle tecniche avanzate per l'impacchettamento e l'interconnessione dei chip, come ad esempio *Multi Chip Modules*. Uno studio di fattibilità preliminare ha mostrato che con una tecnologia MCM sarebbe possibile integrare una matrice di 8×8 chip (per un totale di 16k processori equivalente ad una matrice di 128×128) su un substrato di dimensioni 10 cm $\times$ 10 cm. Questo porterebbe ad avere una matrice di processori di dimensioni paragonabili ai problemi da risolvere in applicazioni reali.

Gli svantaggi principali di una soluzione di questo tipo, al limite della tecnologia attuale, sono i seguenti:

- il costo, legato al progetto dei nuovi circuiti, alla fabbricazione, alla verifica dei chip, e al progetto e verifica dei moduli MCM, è valutato di circa un ordine di grandezza superiore a quello del primo prototipo;
- il problema dovuto alla limitazione nello spazio di indirizzamento di ogni processore, intrinseco dell'architettura, non può essere ovviato facilmente, in quanto le tecnologie VLSI disponibili attualmente non offrono la possibilità di integrare memorie di grandi dimensioni.

4.2 Estensione dello Spazio di Indirizzamento

Il problema della limitata quantità di memoria interna di ogni processore può essere risolto con l'uso di memorie esterne, assegnando un canale a singolo-bit per ogni processore per accedervi, come mostrato in figura 4.1.a. Poichè tutti i processori eseguono la stessa istruzione, tutte le memorie esterne vengono indirizzate contemporaneamente con lo stesso indirizzo, che può essere generato da un controllore esterno.

Lo svantaggio principale di questa estensione esterna dei registri è la necessità di un canale di comunicazione globale verso la memoria con lo stesso parallelismo della matrice dei processori. Questa necessità limita la dimensione della matrice e il parallelismo raggiungibile.

Dal punto di vista architetturale una capiente memoria esterna fornisce un'alternativa elegante alla scansione dei dati in sottofinestre implementata nel-

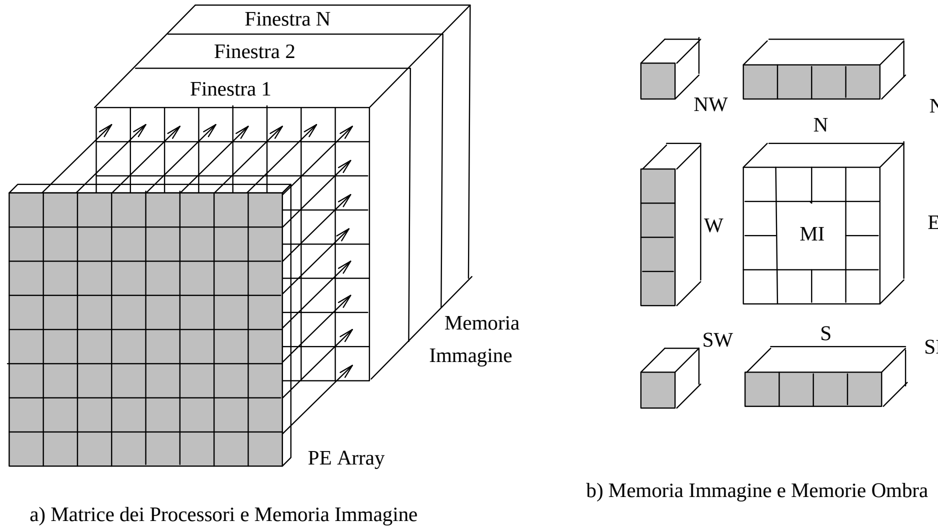


Figura 4.1. Estensione dello spazio di indirizzamento

l'attuale versione di PAPRICA. Questo consentirebbe una *Virtualizzazione Interna* dei processori (vedi capitolo 3.4). L'operazione di scansione diventerebbe quindi solo una operazione di traduzione di indirizzi, o meglio di generazione di offset di memoria.

Sfortunatamente l'utilizzo di operatori morfologici ripropone lo stesso problema dell'elaborazione ai bordi della matrice dei processori già presentato nel capitolo 3.6. Infatti, i processori ai bordi della matrice di elaborazione non hanno accesso al completo vicinato e non possono quindi produrre risultati corretti. Questo problema può essere risolto utilizzando delle memorie aggiuntive, chiamate *Memorie Ombra*, che contengono una copia dei dati memorizzati nelle locazioni del bordo della memoria immagine, come mostrato in figura 4.1.b.

Quando la matrice dei processori elabora una particolare sottofinestra, queste memorie forniscono ai processori del bordo i dati necessari per il completamento del loro vicinato; ognuno di essi, però vi accede con indirizzi diversi, in quanto i dati si riferiscono a sottofinestre differenti. Quando i risultati dell'elaborazione vengono riscritti nella memoria immagine, le memorie ombra vengono preparate per l'elaborazione successiva, e cioè vengono caricate con i valori letti dalla memoria immagine. Questo meccanismo richiede una manipolazione di indirizzi da parte di un circuito esterno dedicato, la cui complessità è estremamente più elevata di quella richiesta dall'architettura corrente per la scansione delle sottofinestre.

Inoltre, poichè i dati risiedono nella memoria esterna, le prestazioni del sistema sono direttamente collegate ai tempi di indirizzamento e di accesso alla memoria, e di conseguenza la velocità della generazione degli indirizzi diventa un punto critico del progetto.

4.3 Evoluzione Architetturale

I problemi incontrati nel tentativo di aumentare le prestazioni del sistema, come mostrato nel capitoli 4.1 e 4.2, sono legati principalmente al meccanismo di virtualizzazione: è infatti sempre necessario dividere l'immagine in sottofinestre e, per poter eseguire operazioni morfologiche, è necessario conoscere lo stato del vicinato dei pixel ai bordi di ogni sottofinestra.

Il meccanismo di virtualizzazione viene notevolmente semplificato se almeno una delle due dimensioni può essere completamente caricata all'interno della matrice dei processori, in quanto in questo caso è necessario scandire l'immagine solo nell'altra dimensione. Questa è la ragione principale per cui è stato avviato lo studio e la valutazione di una nuova soluzione architetturale, in cui i processori sono disposti in modo lineare (monodimensionale) e non più in maniera bidimensionale, e il loro numero è sufficiente da contenere una linea completa dell'immagine da elaborare [87].

L'architettura lineare fornisce prestazioni molto migliori della precedente (in cui i processori erano disposti sui nodi di una griglia bidimensionale) e può quindi essere impiegata in sistemi di visione in tempo reale, dove la velocità di elaborazione e la latenza dei risultati sono i fattori più importanti e critici. Per ottimizzare il sistema completo si sono considerate le seguenti caratteristiche:

- i dati vengono acquisiti in modo seriale dal dispositivo di acquisizione che generalmente scandisce l'immagine colonna per colonna; di conseguenza i dati sono resi disponibili una riga per volta;

- le dimensioni dell'immagine sono fisse e date dalla risoluzione del sensore;
- i risultati della elaborazione di basso livello devono comunque essere elaborati da altri processori per compiti di livello superiore;

In questo quadro, la disposizione dei processori su un vettore monodimensionale è la miglior soluzione che consente di minimizzare la latenza dei dati in quanto una riga può essere processata mentre la successiva viene acquisita dal sensore. Inoltre la risoluzione delle immagini che generalmente vengono elaborate varia da 128 a 1024 pixel: queste dimensioni ridotte rendono possibile la disponibilità di un'architettura con un numero di processori uguale alla dimensione dell'immagine.

In Appdendice E è presentata nel dettaglio l'architettura della nuova versione del sistema PAPRICA, che è attualmente in fase di simulazione.

Capitolo 5

Applicazioni del Sistema PAPRICA

Come già accennato nel capitolo 1, il sistema PAPRICA è stato inizialmente progettato per velocizzare le operazioni di verifica di layout di circuiti integrati. Tali algoritmi operano su strutture dati bidimensionali (immagini in forma iconica).

In seguito il sistema PAPRICA è stato utilizzato come acceleratore hardware anche per altri tipi di problemi, tutti comunque legati all'elaborazione di immagini. Il più importante è senz'altro l'utilizzo del sistema PAPRICA in ambito *automobilistico* a bordo veicolo per l'analisi di immagini in tempo reale, finalizzata all'identificazione della corsia, della carreggiata, di eventuali ostacoli sul percorso, e dell'eventuale tempo di collisione con essi.

Inoltre, le particolari caratteristiche del sistema (architettura SIMD, basso costo di produzione, basso consumo, dimensioni fisiche ridotte) hanno suggerito di valutare anche la possibilità di impiegarlo su *sistemi portatili* per la compressione e decompressione di immagini in tempo reale.

I prossimi due paragrafi presentano l'applicazione più significativa che è stata sviluppata per entrambi i campi applicativi sopra menzionati, e descrivono i vantaggi ottenuti dall'impiego del sistema PAPRICA: la deternimazione della carreggiata per quanto riguarda il campo automobilistico e la compressione di immagini nell'altro settore.

5.1 Assistenza alla Guida

La ricerca in ambito "veicolistico", al fine di incrementare la sicurezza e dare un contributo significativo all'incremento dell'efficienza e dell'utilizzazione delle risorse nel traffico stradale, si articola in diversi aspetti che concorrono a formare un quadro complesso. Tale ricerca ha come scopo ultimo quello di eliminare i punti deboli esistenti nel traffico odierno attraverso la creazione di sistemi "intelligenti" di controllo e di informazione, focalizzandosi sopratutto sull'abilità del veicolo di regolare velocità e traiettoria sulla base di informazioni provenienti da sensori visivi e sulla propria conoscenza sull'ambiente esterno. Se la ricerca si può considerare sufficientemente matura per quanto riguarda alcuni degli aspetti del progetto di sistemi intelligenti, non lo è altrettanto per le metodologie di analisi del moto con sistemi di visione artificiale e per l'applicazione di tecnologie di tipo logico, analogico e neurale alla visione, pianificazione e ragionamento. Vari punti necessitano di ulteriore approfondimento, ad esempio: stima del moto da sequenze di immagini; ricostruzione di scene da viste prospettiche calibrate; soluzione del problema delle corrispondenze; fusione dei dati provenienti da sensori di natura diversa.

Le applicazioni che sono state studiate, concepite e sviluppate per essere integrate sull'architettura PAPRICA vanno dalla identificazione dei bordi strada [16, 24, 25, 20, 19, 15] per l'inseguimento automatico della corsia di marcia, allo sviluppo di tecniche basate su flusso ottico [2, 4] per l'individuazione di ostacoli e per il calcolo del tempo di collisione, fino a studi teorici sulla percezione visiva mirati alla ricostruzione delle occlusioni [3].

I sistemi sviluppati sono stati integrati sul laboratorio mobile MOB-LAB (MOBile LABoratory, riprodotto nella fotografia riportata in figura 5.1), il veicolo dimostratore delle attività delle Unità Accademiche italiane coinvolte nei

Figura 5.1. Il veicolo di prova MOB-LAB: sono visibili due telecamere nella parte anteriore alta

sottoprogetti PRO-ART e PRO-CHIP del progetto Europeo PROMETHEUS. Le funzionalità integrate su MOB-LAB sono state dimostrate al Board Member Meeting tenutosi presso il circuito di prova C.E.R.A.M. di Parigi al termine del progetto, nell'Ottobre 1994.

Nel prossimo paragrafo viene analizzato teoricamente il problema del riconoscimento di caratteristiche mediante elaborazione locale, mentre nel successivo viene discusso un approccio differente, basato su elaborazione multirisoluzione. Infine nell'ultimo paragrafo di questo capitolo viene presentato un approccio basato sulla rimozione dell'effetto prospettico: quest'ultimo ha portato alle migliori prestazioni, sia in termini di qualità del riconoscimento sia in termini di velocità di elaborazione.

Identificazione Bordi Strada: Primo Approccio

Estrazione di caratteristiche mediante elaborazione locale

Il problema della definizione del comportamento del filtro è sostanzialmente ridotto alla definizione delle caratteristiche da evidenziare nell'immagine, partendo da ipotesi che si considerano sempre verificate. In altre parole, ogni pixel deve essere in grado di riconoscere se l'informazione che porta è significativa o meno, cioè se appartiene o meno all'oggetto da evidenziare. Di conseguenza, poichè i dati disponibili ad ogni pixel sono ristretti ad un piccolo intorno del pixel stesso, ogni pixel $p(x, y)$ determina, sulla base della configurazione del proprio vicinato $I(x, y)$, uno Stato Descrittivo $\mathcal{D}(x, y)$ mediante una funzione f, come descritto dalla formula seguente:

$$\mathcal{D}(x, y) = f\Big(p(u, v)\Big) \qquad \text{con } u, v \in I(x, y)\ . \tag{5.1}$$

Ogni pixel, sempre mediante la stessa funzione f, determina un secondo stato, lo stato Caratteristico $\mathcal{C}(x, y)$, assumendo che la configurazione del vicinato sia quella che si ha in presenza dell'oggetto da evidenziare, come descritto nella seguente formula:

$$\mathcal{C}(x, y) = f\Big(x, y, \text{ipotesi}\Big)\ . \tag{5.2}$$

Questi due stati vengono infine comparati; la significatività del pixel è quindi funzione della distanza tra questi due stati e viene codificata nello stato $\mathcal{S}(x, y)$, definito come segue:

$$\mathcal{S}(x, y) = \Big|\mathcal{D}(x, y) - \mathcal{C}(x, y)\Big|\ . \tag{5.3}$$

Per semplificare l'analisi dei risultati, il valore dello stato $\mathcal{S}(x, y)$ può essere eventualmente binarizzato con un meccanismo a soglia. La binarizzazione può essere eseguita anche con un'operazione di isteresi [31], che tende a promuovere come significativi i pixel di poco sottosoglia adiacenti a pixel a loro volta già considerati significativi.

Individuazione dei bordi strada

Come già accennato, il problema principale è identificabile con la definizione della funzione f che associa uno stato caratteristico $\mathcal{C}(x, y)$ e uno stato descrittivo $\mathcal{D}(x, y)$ ad ogni pixel $p(x, y)$ in funzione della configurazione del proprio vicinato $I(x, y)$. Nel caso specifico in esame (determinazione dei bordi strada) l'ipotesi principale che si considera sempre verificata è che le immagini vengano acquisite da un veicolo in marcia lungo un tratto quasi rettilineo di strada. In questa situazione, il bordo strada può essere definito come segue: la coppia di linee che attraversano l'immagine dal basso fino al punto di fuga (Focus Of Expansion, FOE [10]). La presente definizione è sicuramente troppo generale, e la presenza di rumore nell'immagine porta ad un'errata interpretazione delle caratteristiche. Viene considerato rumore, in questo caso, tutto quello che porta dei disturbi nell'interpretazione dell'immagine, ad esempio la tessitura dell'erba e gli alberi (visibili ai bordi e sullo sfondo dell'immagine originale riportata in figura 5.2.a). Per diminuire la possibilità che il rumore influenzi il risultato, alla precedente definizione vengono aggiunti alcuni semplici requisiti: le linee devono avere lunghezza maggiore di uns determinata solgia, e vengono eliminati i punti in cui esse presentano curvatura maggiore di un altro valore prefissato. Infine, poichè la strada si trova nel semipiano inferiore dell'immagine, divisa idealmente dalla linea di orizzonte passante per il punto di fuga, i pixel significativi con ordinata superiore a quella del punto di fuga saranno eliminati. Sfortunatamente questi requisiti aggiuntivi necessitano dell'introduzione di parametri ad hoc per ogni singola immagine, come ad esempio la posizione del punto di fuga (FOE).

Struttura dell'algoritmo

La struttura del filtro è abbastanza semplice: in primo luogo si determinno le discontinuità luminose nell'immagine mediante operazioni di gradiente, soglia, isteresi e scheletrizzazione, seguite da un apposito filtraggio per l'eliminazione dei punti in cui le linee hanno curvatura maggiore di una determinata soglia (vedi figura 5.2.b) In seguito, mediante un primo filtraggio sull'immagine così ottenuta si determina lo stato descrittivo $\mathcal{D}(x,y)$ in ogni pixel (vedi figura 5.2.c); in seguito viene sintetizzato lo stato $\mathcal{C}(x,y)$, corrispondente alla configurazione del vicinato in presenza della caratteristica da evidenziare (vedi figura 5.2.d); al termine viene calcolata la distanza tra questi due stati (rappresentata in figura 5.2.e), seguita da una binarizzazione tramite un filtraggio a soglia o ad isteresi. Inoltre viene aggiunto un passaggio per eliminare le linee più lunghe di una determianta soglia (vedi figura 5.2.f).

Identificazione Bordi Strada: Secondo Approccio

Estrazione di caratteristiche mediante approccio piramidale

La procedura che è stata utilizzata per l'identificazione del bordo strada è basata su euristiche riguardanti la posizione e la forma della carreggiata. Tali informazioni note a priori sono codificate in un'immagine *sintetica* che viene utilizzata dall'algoritmo per determinare eventuali corrispondenze con l'immagine *naturale* acquisita. Il confronto tra le due immagini avviene a risoluzioni basse, beneficiando della possibilità di utilizzare PAPRICA come architettura piramidale, e il risultato viene poi raffinato e riportato alla risoluzione originale.

È facile capire come l'immagine sintetica e l'immagine naturale in ingresso non possano essere direttamente confrontate con tecniche locali; infatti quest'ultima presenta un grado di dettaglio di gran lunga maggiore della prima. Tra i tanti metodi per diminuire la quantità di dettagli è necessario sceglierne uno che garantisca che le caratteristiche di interesse non diminuiscano la loro rilevanza. A tale proposito un semplice filtraggio passa-basso, come una media, diminuendo l'intensità delle discontinuità luminose, ridurrebbe la rilevanza non solo dei dettagli, ma anche dei bordi strada, rendendone più difficoltoso il rilevamento. Al contrario, dato che i bordi strada costituiscono una caratteristica con correlazione a lunga distanza, un sottocampionamento sia dell'immagine naturale che di quella sintetica consente un confronto meno dipendente dai det-

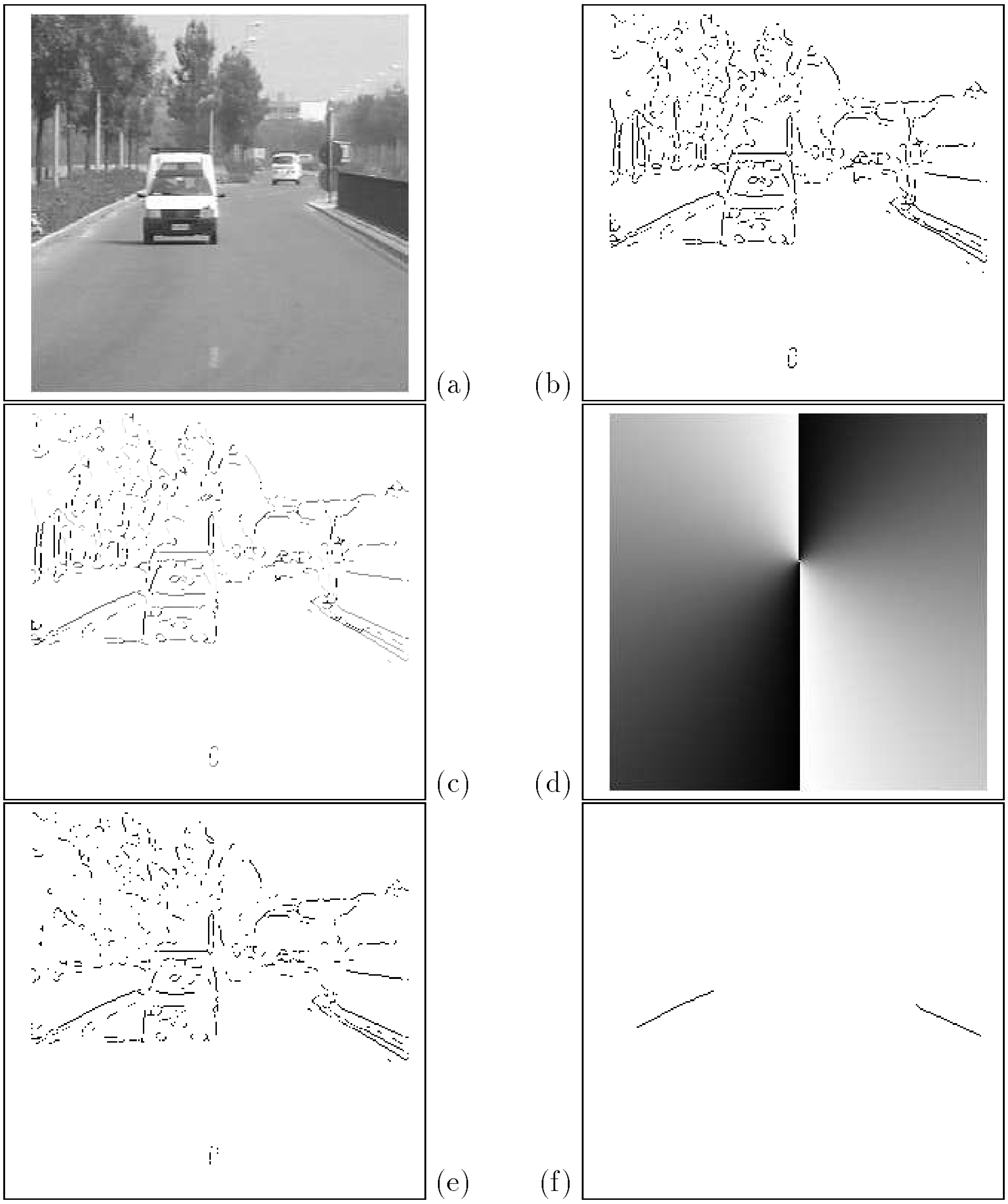


Figura 5.2. Identificazione bordi strada mediante elaborazione locale: *a*) immagine originale; *b*) immagine ottenuta con preelaborazioni quali gradiente, soglia, scheletrizzazione, ed eliminazione dei punti con elevata curvatura locale; *c*) stato descrittivo $\mathcal{D}$; *d*) stato caratteristico $\mathcal{C}$ (immagine sintetica); *e*) significatività dei pixel: a valore più scuro corrisponde maggiore probabilità che il pixel appartenga ad una linea diretta verso il FOE; *f*) eliminazione delle linee più corte di una soglia.

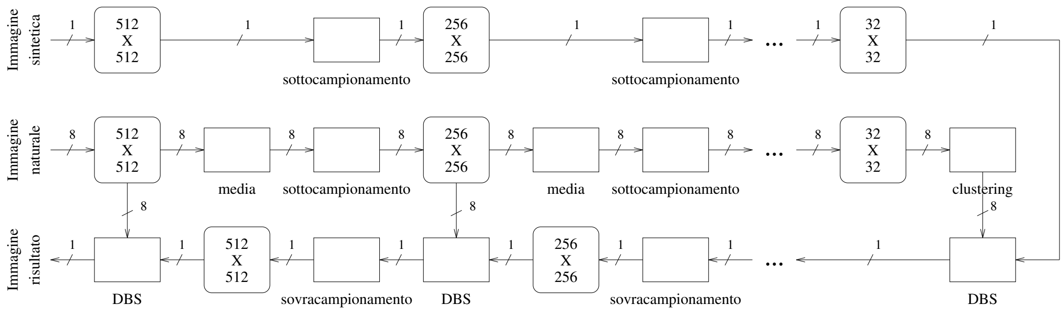


Figura 5.3. Diagramma dell'algoritmo multirisoluzione per l'identificazione dei bordi strada

tagli della prima. In generale, è molto più facile riconoscere oggetti in immagini con bassa risoluzione spaziale rispetto al caso di immagini ad alta risoluzione. Infatti nell'immagine a bassa risoluzione possono non comparire dettagli fuorvianti che invece sono presenti nell'immagine ad alta risoluzione. Comunque, nonostante la rivelazione di un oggetto sia più agevole a bassa risoluzione, in generale una descrizione completa dell'oggetto necessita di dettagli presenti solo alle alte risoluzioni. Queste considerazioni spiegano l'utilità di una struttura dati piramidale, consistente in un'unica immagine a diversi livelli di risoluzione; la ricerca di un oggetto viene iniziata alle basse risoluzioni, ed è via via raffinata mano a mano che si ritorna a risoluzioni maggiori, fino a che non si raggiunge il livello di dettaglio di interesse.

È da notare poi che, nel caso in cui il numero di elementi di elaborazione dell'architettura utilizzata sia inferiore al numero di pixel contenuti nell'immagine, diminuendo la risoluzione diminuisce anche il numero di computazioni da eseguire. Ne segue che la scelta di un paradigma computazionale piramidale, oltre ad essere supportata da ragioni teoriche, risulta essere vantaggiosa anche dal punto di vista dell'efficienza e delle prestazioni.

Struttura dell'algoritmo

Ogniqualvolta si esegue un sottocampionamento, è necessario prefiltrare l'immagine con un filtro passa-basso, come mostrato in figura 5.3; in tale modo si riesce a ridurre l'influenza dei dettagli, nonchè gli effetti dell'*aliasing* [79] dovuto al sottocampionamento.

Tra i vari filtri passa-basso si è pensato di sceglierne uno poco pesante dal punto di vista computazionale e di facile implementazione: la media aritmetica dei valori in un vicinato 8-connesso.

Una volta raggiunta la risoluzione minima (nel caso in figura, 32×32), prima di eseguire il filtraggio DBS l'immagine naturale viene sottoposta ad un'operazione di *clustering* [27], che consiste in una media dei valori in un vicinato 8-connesso pesati dal valore del loro gradiente. In tal modo si riduce l'intensità delle discontinuità luminose meno rilevanti, mentre quelle più significative vengono enfatizzate.

L'immagine sintetica e quella clusterizzata vengono poi confrontate. Il modello teorico della regione della strada codificato nell'immagine sintetica viene modificato mediante un algoritmo iterativo (*Driven Binary Stretching, DBS*) [19] in modo da avvicinarsi il più possibile alla forma della strada presente nell'immagine naturale. Il risultato ottenuto viene quindi riportato a risoluzione superiore e raffinato ulteriormente impiegando sempre lo stesso algoritmo. Tale procedimento viene ripetuto fino ad arrivare alla risoluzione di partenza.

L'algoritmo DBS

In figura 5.4 è rappresentato il filtraggio DBS; l'insieme dei dati in ingresso è costituito da un'immagine binaria (immagine sintetica) e da un'immagine a toni di grigio (immagine naturale). Quest'ultima guida la modifica della prima, generando una nuova immagine binaria mediante tecniche locali.

Lo scopo dell'algoritmo è quello di individuare nell'immagine naturale i bordi della strada. Generalmente al contorno di un oggetto sono associate variazioni dell'intensità luminosa piuttosto rilevanti; ne segue che un primo passo per l'estrazione dei bordi strada consiste nell'individuare nell'immagine naturale le discontinuità dell'intensità luminosa più significative. Infatti in primo luogo l'algoritmo DBS prevede il calcolo del gradiente dell' immagine naturale, seguito da un'operazione di soglia, in modo da ottenere un'immagine binaria contenente solo le variazioni di intensità luminosa più significative.

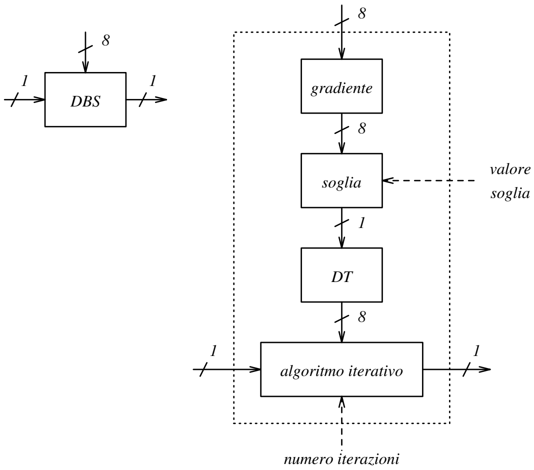


Figura 5.4. Diagramma dell'algoritmo DBS

Lo scopo dell'algoritmo presentato nel seguito è quello di sagomare l'immagine sintetica in modo che segua il più possibile l'andamento dei pixel di *foreground* dell'immagine binaria: ogni pixel del contorno dell'immagine sintetica viene spostato in una direzione che riduce una determinata *funzione costo*. In un punto la funzione costo è rappresentata dalla minima distanza tra il punto e i punti per i quali il gradiente è maggiore di una determinata soglia, ovvero i punti di *foreground*. Per tale motivo come secondo passo dell'algoritmo DBS si calcola la Trasformata Distanza (*Distance Transform, DT*) [12] dell'immagine gradiente binarizzata. La trasformata distanza fornisce punto per punto la distanza dai pixel di *foreground* dell'immagine binaria. Per spiegare cosa si intende per Trasformata Distanza occorre introdurre il concetto di *metrica*. Per metrica si intende una funzione $d : E \times E \to E$ per la quale sono valide le seguenti proprietà :

- $d(p,p) = 0$
- $d(p,q) = d(q,p)$

- $d(p, q) + d(q, s) \geq d(p, s)$

con $p, q, s \in E$.

Nel caso in questione si è adottata la seguente metrica (*city-block*):

$$d(p, q) = |x_p - x_q| + |y_p - y_q| \ , \tag{5.4}$$

dove $p = (x_p, y_p)$ e $q = (x_q, y_q)$. La figura 5.5 mostra le distanze che tale metrica produce dal punto nell'origine.

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Figura 5.5. Metrica *city-block*

Indicando con f_i i pixel di *foreground* il valore della trasformata distanza $\mathcal{D}$ nel generico punto p è dato dalla seguente relazione:

$$\mathcal{D}(p) = \min_i d(p, f_i) \ . \tag{5.5}$$

In realtà nel seguito del lavoro si è considerato il complemento a λ della trasformata distanza sopra definita, ovvero

$$\mathcal{D}(p) = \lambda - \min_i d(p, f_i) \ , \tag{5.6}$$

dove λ rappresenta il masssimo valore consentito dalla struttura dati utilizzata (nel caso specifico λ vale 255, poichè si è utilizzato un byte per ogni pixel dell'immagine).

Va osservato che una metrica di questo tipo è efficientemente calcolabile su architetture parallele in cui ogni processore è connesso ai suoi 4 primi vicini. Una trattazione più completa sui metodi con cui calcolare la Trasformata Distanza è riportata in [18].

Siccome l'informazione distanza serve solo per quei punti che appartengono al bordo dell'immagine sintetica e ai loro primi vicini, il processo iterativo può essere fermato quando per tali punti è già stata calcolata (o, nel caso il numero di iterazioni sia troppo alto, fino ad una certa soglia). Di conseguenza l'andamento della DT nelle zone di interesse non è mai costante se non nelle zone di massimo.

Il cuore dell'algoritmo DBS è costituito da un algoritmo iterativo, il cui scopo è quello di muovere i punti del contorno dell'immagine sintetica lungo le direzioni in cui la DT cresce, ovvero verso i punti in cui la DT è massima. Nella figura 5.6 è mostrato un esempio relativamente al caso monodimensionale.

Tra le varie soluzioni possibili al problema se n'è scelta una che si rifà ai concetti della morfologia matematica. Tale soluzione prevede l'esecuzione in tempi diversi di un'espansione e di un'erosione morfologica impiegando un vicinato 4-connesso, e di tenere validi i risultati di tali operazioni solo per quei punti del contorno dell'immagine sintetica che si sono spostati nelle direzioni in cui la DT è crescente.

Gli elementi generati dall'espansione di S con l'elemento strutturante 4-connesso:

$$N = \Big\{(-1,0);(1,0);(0,0);(0,-1);(0,1)\Big\} \tag{5.7}$$

costituiscono il *bordo esterno* di S, che si detetermina dalla seguente relazione:

$$\mathcal{B}_e(S) = (S \oplus N) \cap \overline{S} \tag{5.8}$$

Analogamente i punti rimossi dall'erosione di S con N, costituiscono il *bordo interno* di S:

$$\mathcal{B}_i(S) = \overline{(S \ominus N)} \cap S \tag{5.9}$$

L'insieme degli elementi di $\mathcal{B}_e(S)$ e di $\mathcal{B}_i(S)$ costituisce il *bordo* di S:

$$\mathcal{B}(S) = \mathcal{B}_e(S) \cup \mathcal{B}_i(S) \tag{5.10}$$

Tali elementi sono gli unici che vengono modificati durante una singola iterazione dell'algoritmo.

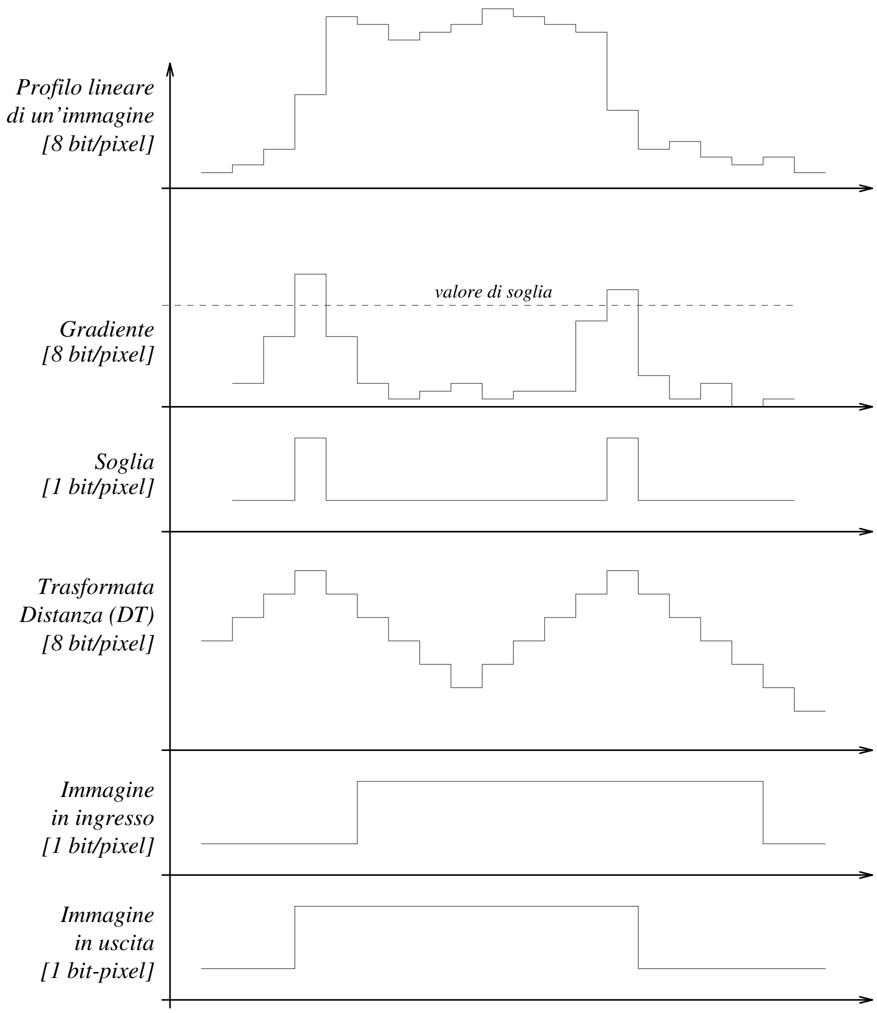


Figura 5.6. Esempio di funzionamento dell'algoritmo DBS (4 iterazioni)

Fatte queste premesse, l'algoritmo in questione può essere formalizzato nel seguente modo:

Regola per il bordo esterno:

- si calcola per tutti i pixel del bordo esterno il valore minimo della DT per tutti i punti di un vicinato 4-connesso che appartengono all'insieme S;
- si inseriscono nell'insieme S (si settano) tutti quei pixel del bordo esterno

per i quali la DT assume un valore maggiore del minimo precedentemente calcolato.

Regola per il bordo interno:

- si calcola per tutti i pixel del bordo interno il valore minimo della DT considerando i punti di un vicinato 4-connesso che non appartengono all'insieme S;
- si eliminano dall'insieme S (si resettano) tutti quei pixel del bordo esterno per i quali la DT assume un valore maggiore del minimo precedentemente calcolato.

Si analizza ora la regola per il bordo esterno, considerando il passo n dell'algoritmo iterativo.

Si definisce l'operazione $\odot : E^2 \times E^3 \rightarrow E^3$ di *mascheratura pixel per pixel* tra un'immagine binaria A ed una a toni di grigio B:

$$A \odot B \triangleq \{c \in E^3 \mid c = (u, v) \in B,\ u \in A\} \tag{5.11}$$

$A \odot B$ è l'insieme degli elementi contenuti in A con valore specificato in B. Si considera perciò l'immagine

$$K_e^{(n)} = (S_e^{(n)} \odot DT) \cup (\overline{S_e^{(n)}} \odot L) \tag{5.12}$$

dove $S_e^{(n)}$ rappresenta l'immagine S_e al passo n; il pedice e indica che si sta considerando il caso del bordo esterno e l'insieme L è così definito:

$$L = \{l \in E^3 \mid l = (u, \lambda),\ \forall\, u \in E^2\} \tag{5.13}$$

dove λ, come accennato in precedenza, rappresenta il massimo valore consentito nella scala dei livelli di grigio.

L'operazione di assegnamento ad un elemento dell'insieme $K_e^{(n)}$ del minimo valore associato agli elementi del suo vicinato 4-connesso coincide con la seguente erosione, come giustificato in [61]:

$$M_e^{(n)} = K_e^{(n)} \ominus Q \tag{5.14}$$

dove

$$Q = \{(1,0,0);\ (-1,0,0);\ (0,1,0);\ (0,-1,0)\} \tag{5.15}$$

come mostrato in figura 5.7. Si determina poi l'insieme dei pixel in cui il valore

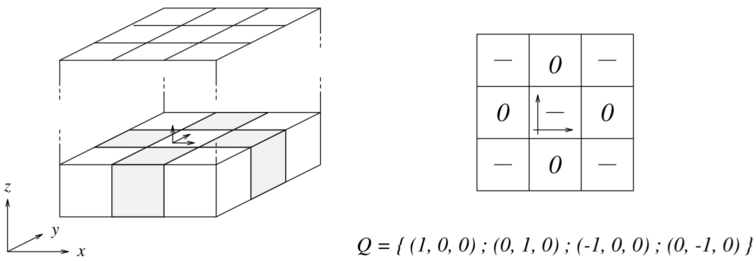


Figura 5.7. Elemento strutturante Q

di $M_e^{(n)}$ è minore del valore di DT. Sia $\mathcal{M} : E^3 \times E^3 \to E^2$ una funzione tale per cui

$$\mathcal{M}(A,B) \triangleq \{x \in E^2 \mid \mathcal{V}(A,x) < \mathcal{V}(B,x)\} \tag{5.16}$$

dove $\mathcal{V} : E^3 \times E^2 \to E$ è così definita:

$$\mathcal{V}(A,x) \triangleq \begin{cases} a & \text{se } \exists\, a \in E \mid (x,a) \in T(A) \\ 0 & \text{altrimenti} \end{cases} \tag{5.17}$$

dove $T(A)$ rappresenta il *Top* di A. L'insieme dei pixel cercato è dato da $\mathcal{M}(M, DT)$. In esso si considerano solo i pixel che appartengono al bordo esterno di S.

$$E_e^{(n)} = \mathcal{M}(M_e^{(n)}, D) \cap \mathcal{B}_e(S_e^{(n)}) \tag{5.18}$$

Il risultato finale è dato dalla seguente relazione:

$$S_e^{(n+1)} = S_e^{(n)} \cup E_e^{(n)} \tag{5.19}$$

In figura 5.8 è rappresentato l'esecuzione di tale regola relativamente al caso monodimensionale.

Con passaggi analoghi si implementa la regola per il bordo interno; si considera perciò l'immagine

$$K_i^{(n)} = \overline{S_i^{(n)}} \odot DT \tag{5.20}$$

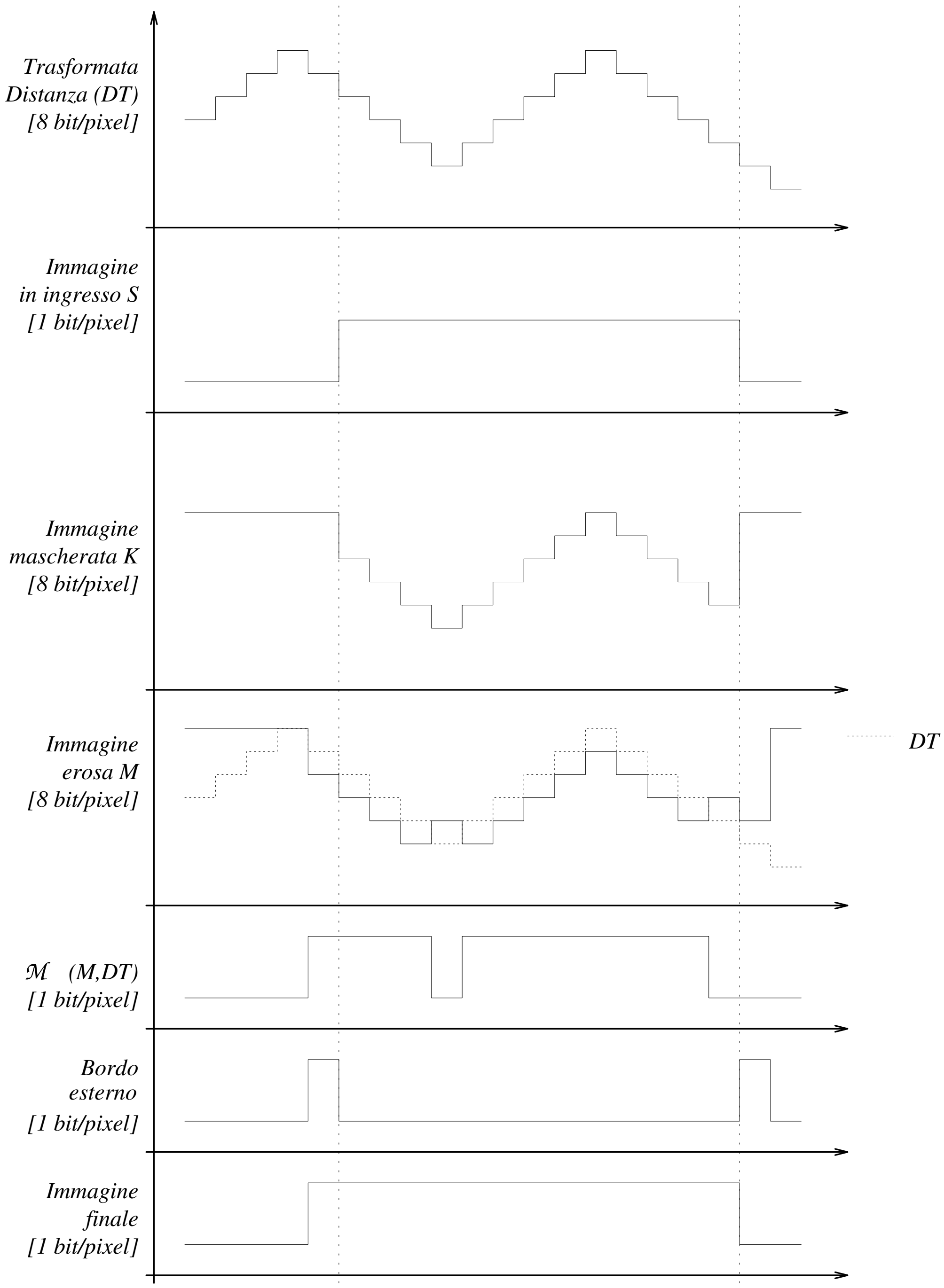


Figura 5.8. Elaborazione nel caso di Bordo Esterno

dove il pedice i sta ad indicare che si considera il bordo interno. L'operazione di assegnamento ad un elemento dell'insieme $K_i^{(n)}$ del massimo valore associato agli elementi del suo vicinato 4-connesso coincide con la seguente espansione:

$$M_i^{(n)} = K_i^{(n)} \oplus Q \tag{5.21}$$

L'insieme dei pixel in cui il valore di $M_i^{(n)}$ è minore del valore di DT è dato da $\mathcal{M}(M_i^{(n)}, DT)$. In esso si considerano solo i pixel che appartengono al bordo interno di $S_i^{(n)}$.

$$E_i^{(n)} = \mathcal{M}(M_i^{(n)}, DT) \cap \mathcal{B}_i(S_i^{(n)}) \tag{5.22}$$

Il risultato finale è dato dalla seguente relazione:

$$S_i^{(n+1)} = \overline{(\overline{S_i^{(n)}} \cup E_i^{(n)})} = S_i^{(n)} \cap \overline{E_i^{(n)}} \tag{5.23}$$

In figura 5.9 è mostrata l'esecuzione della regola per il bordo interno nel caso monodimensionale.

Le figure 5.10 e 5.11 mostrano rispettivamente l'identificazione della carreggiata e della corsia; la differenziazione avviene mediante la scelta di due immagini sintetiche differenti. Le due immagini sintetiche utilizzate nei due casi riportati nelle figure 5.10 e 5.11 sono rappresentate in figura 5.12.

Identificazione Bordi Strada: Terzo Approccio

Ricampionamento dell'immagine e rimozione dell'effetto prospettico

Grazie alla possibilità di ottenere alte prestazioni, la porzione di basso livello dell'elaborazione delle immagini è generalmente demandata a processori SIMD che utilizzano paradigmi computazionali massivamente paralleli (come ad esempio Automi Cellulari o Morfologia Matematica). Questo significa che ogni pixel viene elaborato simultaneamente e utilizzando la stessa regola elaborativa.

Nonostante questo, un approccio massivamente parallelo è significativo quando l'immagine è considerata come un insieme di pixel, indipendentemente dal

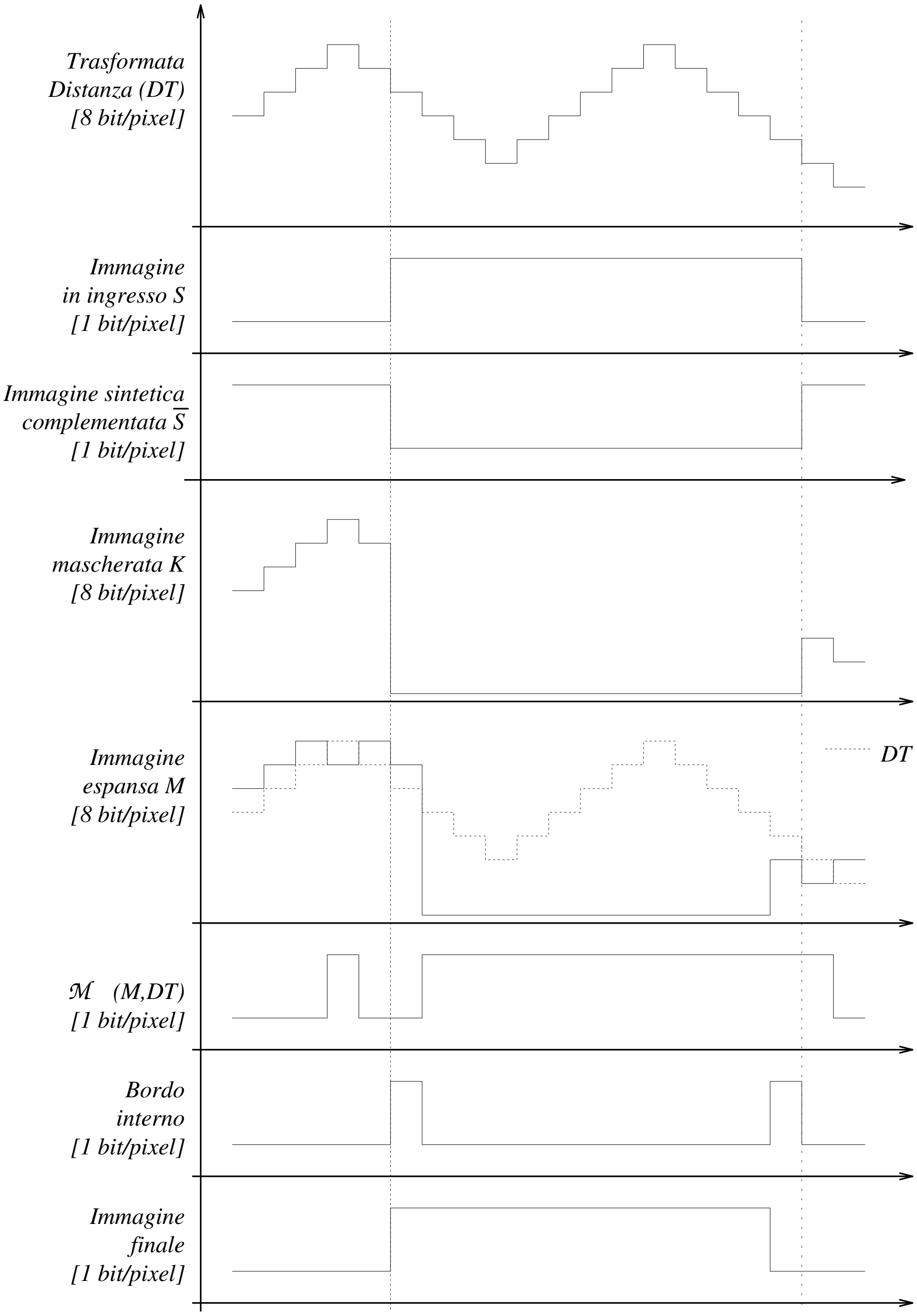


Figura 5.9. Elaborazione nel caso di Bordo Interno

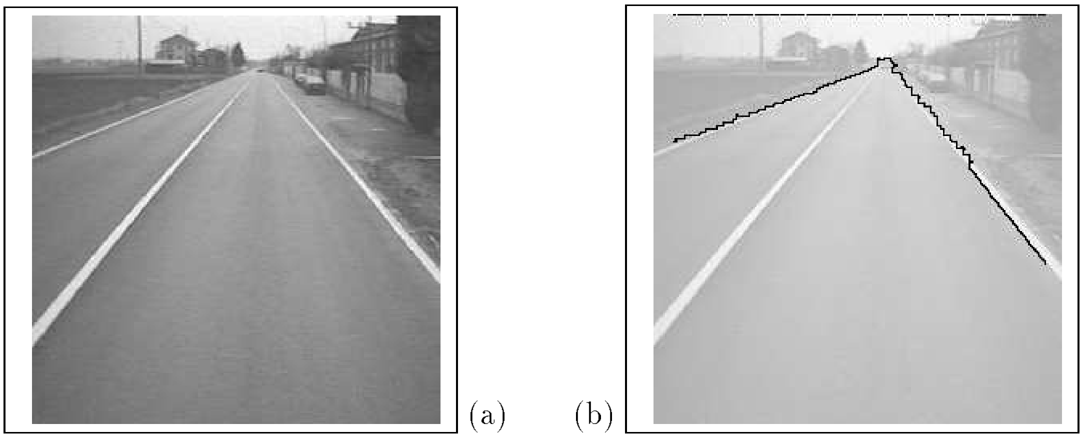


Figura 5.10. Identificazione bordi strada mediante ricerca guidata dalle aspettative: *a*) immagine originale; *b*) immagine ottenuta mediante la sovrapposizione del risultato della deformazione dell'immagine sintetica sull'originale.

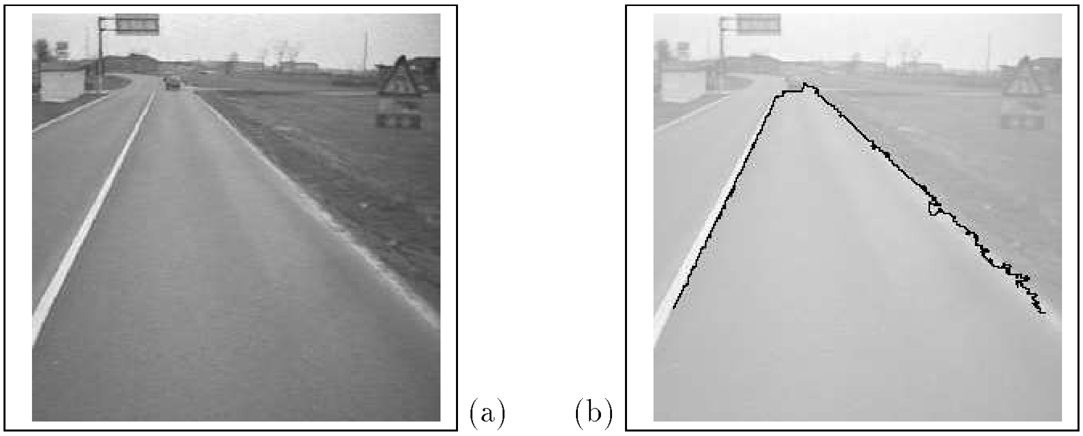


Figura 5.11. Identificazione bordi carreggiata mediante ricerca guidata dalle aspettative: *a*) immagine originale; *b*) immagine ottenuta mediante la sovrapposizione del risultato della deformazione dell'immagine sintetica sull'originale.

suo contenuto semantico (come per esempio per il filtraggio del rumore, l'aumento del contrasto,... come mostrato in figura 5.13). Viceversa, se è richiesta un'elaborazione più complessa (come per esempio l'individuazione dei bordi strada), perfino la porzione di basso livello deve essere guidata da conoscenza a priori sul tipo di caratteristiche da evidenziare, e sul contenuto semantico atteso

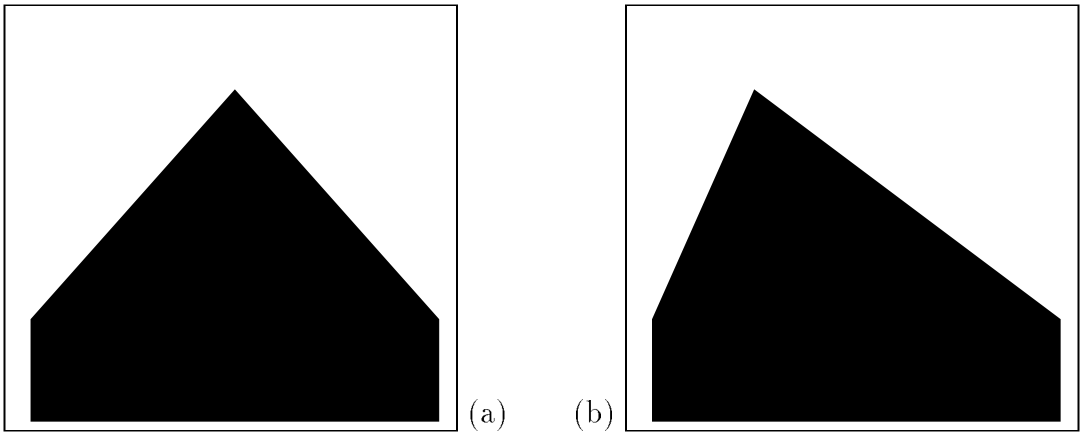


Figura 5.12. Immagini sintetiche utilizzate per l'identificazione a) della carreggiata, e b) della corsia.

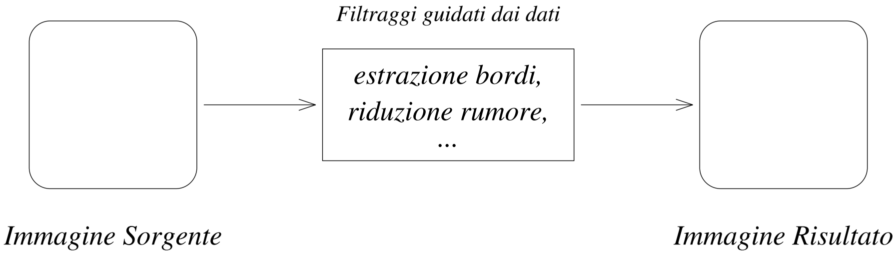


Figura 5.13. Filtraggi guidati dai dati (*data-driven*)

dell'immagine. Questo è generalmente ottenuto differenziando l'elaborazione in funzione della regione dell'immagine da elaborare (si veda la figura 5.14).

Come esempio si consideri il problema specifico dell'identificazione dei bordi strada, che viene qui ridotto alla determinazione della segnaletica stradale orizzontale. A causa dell'effetto prospettico indotto dalle particolari condizioni di acquisizione, la larghezza delle linee sulla carreggiata rappresentate nell'immagine acquisita varia in funzione dalla loro distanza dalla telecamera, come mostrato in figura 5.18.a, che mostra un'immagine acquisita dal veicolo MOB-LAB. La loro larghezza variabile suggerisce di eseguire elaborazioni diverse in zone diverse dell'immagine, ma sfortunatamente questa elaborazione differenziata non può essere efficientemente implementata su architetture SIMD massivamente parallele, in quanto per definizione esse eseguono *la stessa* operazione

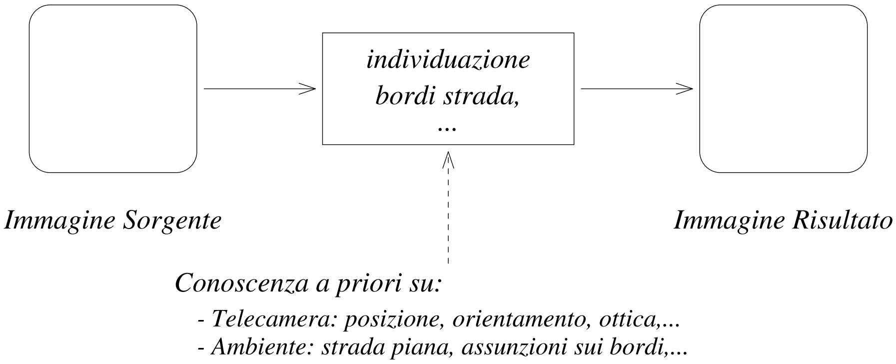


Figura 5.14. L'utilizzo di conoscenza a priori

su ogni pixel dell'immagine.

L'effetto prospettico fa si che i pixel abbiano significati diversi (e quindi proprietà differenti) a seconda della loro posizione nell'immagine. Per esempio, un pixel nella parte bassa dell'immagine mostrata in figura 5.18.a rappresenta alcuni centimetri quadrati della strada, mentre un pixel nella parte centrale della stessa immagine rappresenta alcune decine di centimetri quadrati.

Contrariamente, la rimozione dell'effetto prospettico fa in modo che ogni pixel rappresenti la stessa porzione della strada. In questo modo la larghezza delle linee sulla strada non dipende dalla loro posizione nell'immagine e quindi anche la procedura utilizzata per la loro identificazione è indipendente dalla posizione dei pixel nell'immagine stessa.

Per rimuovere l'effetto prospettico, è necessario conoscere le particolari condizioni di acquisizione (posizione della telecamera sul veicolo, orientamento della stessa, ottiche installate,...) e la scena rappresentata nell'immagine (una strada, che d'ora in poi verrà supposta piana, senza dossi o cunette). Questo costituisce la conoscenza a priori.

In questo modo la procedura di identificazione delle linee sulla strada può essere convenientemente suddivisa in due passi, come mostrato in figura 5.15: il primo (guidato dalle aspettative), sfrutta la conoscenza a priori ed è finalizzato alla rimozione dell'effetto prospettico; il secondo (guidato dai dati) è basato sull'elaborazione di basso livello dei dati sensoriali.

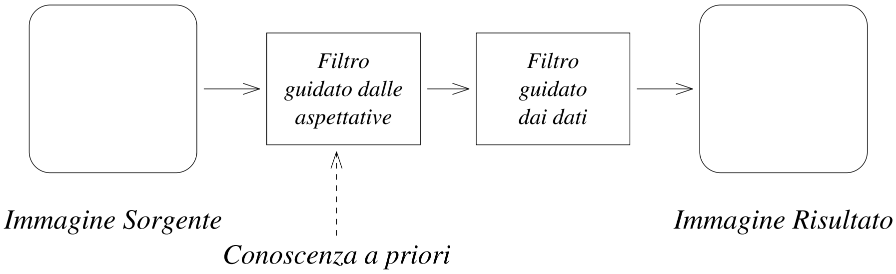


Figura 5.15. L'elaborazione è suddivisa in due passi, dove solo il primo sfrutta la conoscenza a priori

Il vantaggio di questa suddivisione è dato dalla possibilità di implementare il secondo passo in modo estremamente efficiente su architetture massivamente parallele SIMD, e quindi di ottenere elevate prestazioni.

Il processo di riorganizzazione dell'immagine

Come mostrato in figura 5.16, il processo di riorganizzazione dell'immagine può essere considerato alla stregua di un prefiltraggio ed eseguito subito dopo l'acquisizione e prima della tradizionale elaborazione di basso livello. Il sistema preposto alla riorganizzazione ricampiona l'immagine in ingresso replicando, eliminando, o spostando i pixel in diverse posizioni nell'immagine. In questo modo produce una nuova matrice di pixel (immagine in forma iconica).

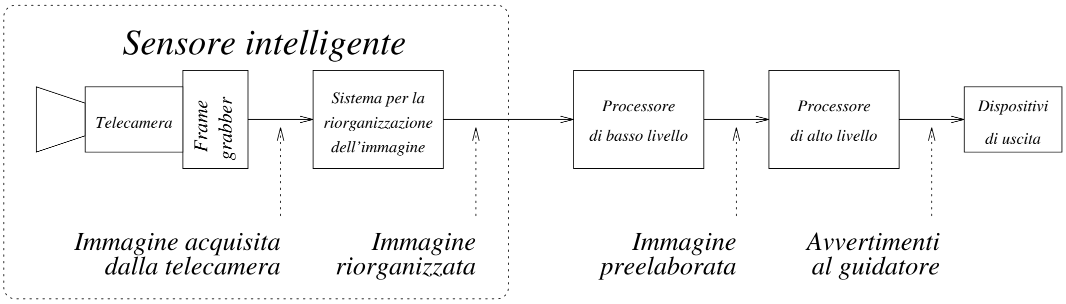


Figura 5.16. Diagramma a blocchi del sistema completo

Vengono definiti due spazi Euclidei:

- $\mathcal{W} \in E^3$ che definisce lo spazio *mondo* a tre dimensioni (*world-coordinate* *system*), dove è rappresentata la scena reale;

- $\mathcal{I} \in E^2$ che definisce lo spazio *immagine* a due dimensioni (*screen-coordinate system*), dove viene proiettata la scena reale a tre dimensioni.

L'immagine originale acquisita dalla telecamera appartiene allo spazio $\mathcal{I}$, mentre l'immagine riorganizzata è definita come il piano $z = 0$ dello spazio $\mathcal{W}$ (sempre supponendo che la strada sia piana). La corrispondenza tra le due immagini è espressa dalle due trasformate duali f e g:

$$\begin{cases} f: & (u,v) \in \mathcal{I} & \rightarrow & (x,y,0) \in \mathcal{W} \\ g: & (x,y,0) \in \mathcal{W} & \rightarrow & (u,v) \in \mathcal{I} \end{cases} , \tag{5.24}$$

che, come descritto in [76], dipendono da:

- *il punto di vista* (la posizione della telecamera);
- *la direzione di vista* (l'orientamento della telecamera);
- *l'apertura angolare* della telecamera;
- *la risoluzione* della telecamera.

La figura 5.17 mostra la relazione tra i due spazi $\mathcal{W}$ e $\mathcal{I}$, mentre in Appendice F è riportata la definizione matematica delle trasformate f e g.

Per definizione, l'immagine riorganizzata rappresenta la porzione di strada di fronte al veicolo come se fosse osservata dall'alto. La figura 5.18.b mostra il risultato della riorganizzazione dell'immagine originale riportata in figura 5.18.a, ottenuta utilizzando i parametri reali [99] della telecamera installata su MOB-LAB.

Il processo di riorganizzazione associa ad ogni pixel $P = (x,y,0)$ dell'immagine riorganizzata il proprio pixel corrispondente $Q = (u,v)$ nell'immagine originale. Una volta ottenuta l'immagine riorganizzata, l'identificazione delle linee sulla strada è un compito facilmente risolubile (vedi Appendice F). Oltre ad essere facilmente implementabile su qualunque architettura massivamente

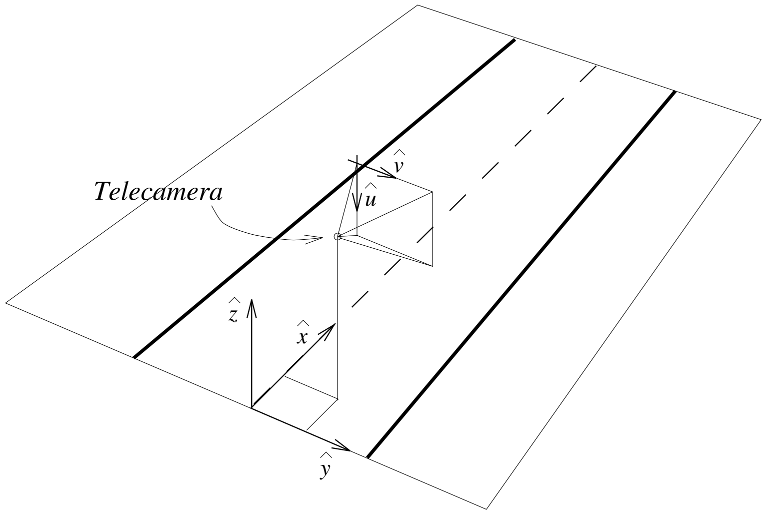


Figura 5.17. La relazione tra i due spazi $\mathcal{W}$ e $\mathcal{I}$

parallela SIMD, il processo di identificazione delle linee [15] ha un grosso vantaggio rispetto agli altri approcci descritti precedentemente. La riorganizzazione dell'immagine risolve il principale problema dei sistemi di visione utilizzati nel campo automobilistico per l'inseguimento della carreggiata (*automotive road-following systems* [54, 78, 66, 64, 65, 95]): la presenza di ombre sulla carreggiata. Grazie alla riorganizzazione, le linee della strada sono rappresentate da linee quasi verticali di larghezza costante, che possono essere facilmente individuate ed estratte attraverso semplici filtraggi morfologici. Il risultato di questa elaborazione è presentato in figura 5.18.c, mentre la figura 5.18.d mostra la sovrapposizione dell'immagine originale con l'immagine risultato in cui è stato nuovamente inserito l'effetto prospettico.

In Appendice F sono riportati alcuni risultati dell'elaborazione di immagini acquisite in condizioni diverse (strada curva, ombre sulla carreggiata, altri veicoli di fronte alla telecamera,...).

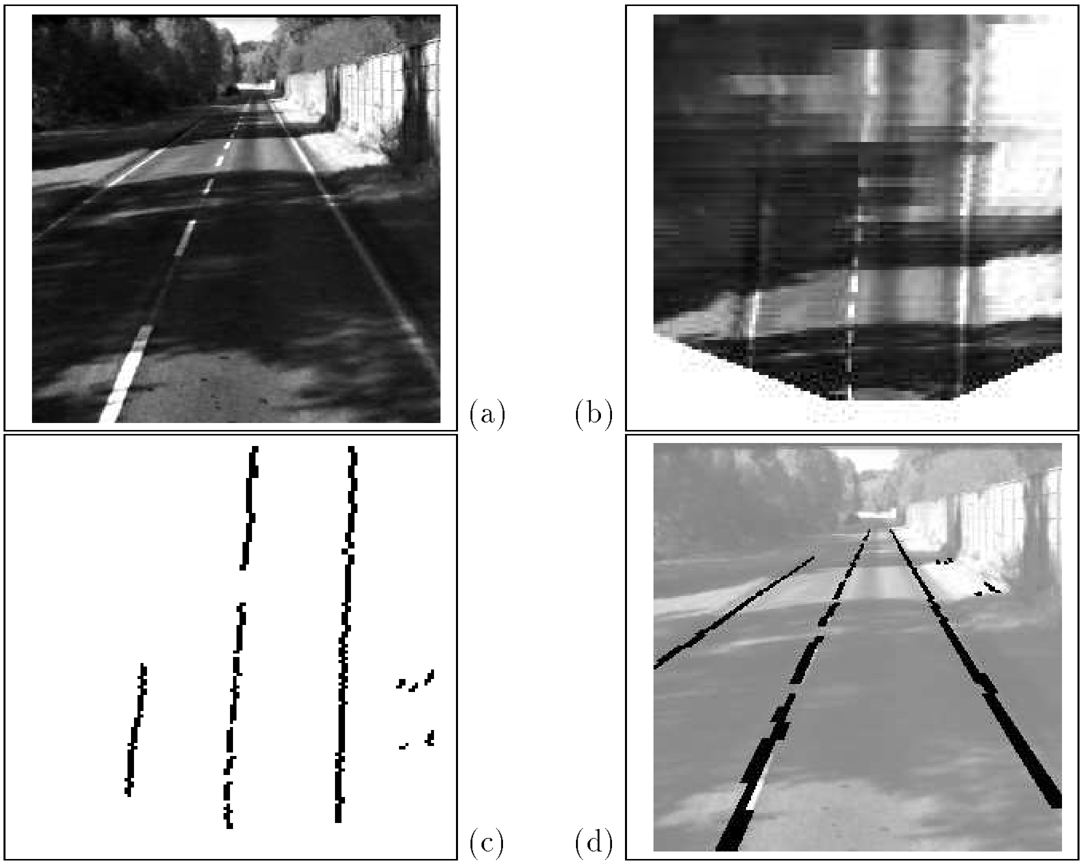


Figura 5.18. Identificazione bordi strada mediante rimozione dell'effetto prospettico: *a*) immagine originale; *b*) immagine ottenuta mediante la rimozione dell'effetto prospettico; *c*) individuazione delle linee sull'asfalto; *d*) sovrapposizione dell'immagine originale e dell'immagine precedente in cui è stato di nuovo inserito l'effetto prospettico.

Confronto tra i Diversi Approcci

Le prestazioni ottenibili con i tre diversi approcci presentati nei precedenti paragrafi sono diverse sia in termini di qualità dei risultati, sia per i tempi di esecuzione.

Il primo approccio è basato esclusivamente sull'elaborazione dell'immagine con tecniche locali. L'unico suo vantaggio è l'estrema facilità di integrazione su architetture massivamente parallele con una topologia di interconnessione tra processori a griglia bidimensionale. Il problema principale è comunque dovuto alla difficoltà di propagare a lunga distanza le informazioni: tale possibilità renderebbe più agevole l'identificazione di caratteristiche di grandi dimensioni come i bordi strada.

Per questo motivo è stato studiato un nuovo approccio, questa volta basato su elaborazioni multirisoluzione. Tali tecniche permettono di ottenere una prima approssimazione del risultato utilizzando una versione a bassa risoluzione dell'immagine originale; questo risultato preliminare viene poi raffinato, passando ad elaborare versioni dell'immagine a risoluzioni sempre maggiori. Un vantaggio notevole di questo approccio piramidale consiste nell'incremento della velocità di elaborazione, in quanto parte di essa viene eseguita su immagini con un minor numero di pixel. Sfortunatamente però, questo approccio, come anche il precedente, ha un grosso problema dovuto al fatto che ogni pixel viene elaborato mediante lo stesso algoritmo, indipendentemente dalla propria posizione all'interno dell'immagine.

Questa è la ragione principale per cui si è progettato un terzo approccio, in cui viene dato un peso diverso ai pixel a seconda della loro posizione nell'immagine. Il motivo di tale differenziazione è legato al diverso significato che assumono i pixel in dipendenza dalle particolari condizioni di acquisizione e in funzione del contenuto semantico della scena rappresentata nell'immagine. Il processo di riorganizzazione[1] consente di uniformare le proprietà dei pixel e di eseguire efficientemente l'elaborazione successiva su architetture SIMD.

Tutti e tre gli approcci descritti sono stati implementati sul sistema PAPRICA direttamente in linguaggio Assembly oppure mediante delle procedure automatiche per la generazione del codice in modo parametrico.

- Nel primo caso, a causa dell'elevato numero di iterazioni necessarie per propagare a lunga distanza le informazioni, l'elaborazione di una singola immagine di dimensioni 256×256 pixel richiede [16] circa $1.4 \div 1.5$ secondi;

[1] Attualmente è in fase di progetto un'estensione architetturale per il secondo prototipo che consente di riorganizzare l'immagine in modo molto efficiente.

- nel secondo caso [19], invece, il numero di tali iterazioni viene ridotto grazie all'approccio piramidale, e l'elaborazione di una singola immagine 256×256 impiega circa 500 ms;

- viceversa, per i motivi descritti sopra la velocità computazionale più elevata si ottiene nell'ultimo caso, dove l'elaborazione di una singola immagine impiega [15] circa 100 ms, rendendo possibile raggiungere una frequenza elaborativa di circa 10 immagini al secondo. Inoltre il terzo approccio, come mostrato nelle figure F.3, F.4, F.5, F.6 e F.7 riportate nell'Appendice F, è quello che fornisce i risultati migliori anche in condizioni proibitive, dove cioè le linee sulla carreggiata sono immerse in zone di chiaroscuro dovute alla presenza di ombre.

5.2 Compressione di Immagini

Le tecniche di analisi del segnale televisivo nel campo dell'alta definizione rappresentano un problema tecnico-scientifico di grande interesse che coinvolge contemporaneamente aspetti algoritmici, architetturali e tecnologici. Mentre i DSP sono stati usati tradizionalmente per l'elaborazione del segnale audio, nel campo dell'elaborazione di immagini dominano i circuiti altamente specializzati. Mentre in applicazioni di tipo audio il rapporto tra la frequenza di clock e la frequenza di campionamento è dell'ordine di 1000 (40 MHz su 40 kHz) nel caso dell'elaborazione del segnale video tale rapporto scende a 1 o meno, con frequenze di campionamento anche superiori a 100 MHz.

Assumendo poi, che l'elaborazione di segnali HDTV richieda l'esecuzione di più di 100 operazioni per campione ad una velocità di campionamento di 100 MHz è chiaro che le prestazioni computazionali dovranno essere dell'ordine di 10 GOPS (10^{12} operazioni al secondo). Questo livello di prestazioni può essere raggiunto solamente attraverso l'uso di architetture di elaborazione parallele, per le quali si pone il problema di massimizzare le prestazioni, tenendo conto dei vincoli rappresentati dall'algoritmo da implementare e dal consumo.

L'obiettivo di sviluppare sistemi HDTV a costo accettabile richiede un attento esame degli aspetti legati all'architettura. La soluzione analizzata in questo lavoro è quella attualmente utilizzata nei sistemi di supercalcolo per ragioni simili (diminuzione dei costi mediante diminuzione della frequenza di clock). L'aspetto più significativo del problema scientifico è quello di accoppiare i possibili algoritmi alle architetture in modo da massimizzare le prestazioni.

In questo lavoro sono state considerate esclusivamente tecniche di tipo "lossy". Sono stati considerati preliminarmente gli algoritmi standard per la compressione del segnale televisivo e sono state poi indagate tecniche innovative finalizzate all'utilizzo di architetture massivamente parallele. L'obiettivo principale è quello di mettere a punto algoritmi che consentano una veloce compressione/decompressione del segnale televisivo e contemporaneamente siano implementabili su architetture con tecnologia non particolarmente spinta. In particolare è stato sviluppato e messo a punto un algoritmo massivamente parallelo (descritto in Appendice G) capace di ottenere alti fattori di compressione e che degrada le immagini in modo poco fastidioso per l'osservatore umano. Tale algoritmo è stato dapprima implementato su Connection Machine CM-2 per facilitare la verifica dei risultati, ed è attualmente in corso di integrazione sull'architettura finale PAPRICA. Di conseguenza l'algoritmo è stato concepito in modo da non necessitare nè di operazioni in virgola mobile (come ad esempio lo standard JPEG), nè di tecniche seriali quali la compressione Lempel-Ziv (implementata anche nel formato GIF). L'algoritmo è basato sull'analisi dell'immagine a differenti risoluzioni. Tale tecnica può essere facilmente implementata anche in hardware mediante tecniche piramidali. L'intera immagine viene suddivisa in blocchi di dimensione due per due. Per ogni blocco viene considerato il valore di un punto di riferimento e viene confrontato con i suoi primi vicini lungo le direzioni orizzontale (x), verticale (y), diagonale (xy). Il valore memorizzato in questi punti viene sostituito con la differenza rispetto al punto di riferimento. Successivamente viene applicata una soglia ai valori così ottenuti, eliminando così i punti con correlazione spaziale più alta. Questo schema, memorizzando in modo oppurtuno i valori dei pixel esaminati, permette di operare in parallelo su tutti i blocchi dell'immagine senza la necessità di comunicazioni di tipo globale. Iterando questo procedimento con tecniche di tipo piramidale si possono ottenere compressioni maggiori. Per il particolare algoritmo impiegato, le operazioni di trasformata e di selezione utilizzano unicamente numeri interi.

L'algoritmo è discusso nei dettagli nell'Appendice G.

Conclusioni

In questo lavoro è stata presentata la prima versione del sistema PAPRICA e sono stati analizzati i suoi principali colli di bottiglia; inoltre sono stati evidenziati i vantaggi che derivano dall'impiego di tale sistema in diversi settori applicativi.

A partire dall'analisi teorica e sperimentale delle prestazioni sono state descritte tre possibili strategie per l'evoluzione del sistema. Fra esse, quella che permette di raggiungere le migliori prestazioni in termini di velocità elaborativa e che allo stesso tempo non richiede tecnologie particolarmente spinte prevede una modifica della topologia di interconnessione tra i processori: da una topologia a griglia bidimensionale si passa ad un vettore monodimensionale. Come conseguenza di questo fatto, la virtualizzazione basata sulla scansione a finestre viene sostituita da una scansione per linee. Inoltre, da un vicinato 3×3 si passa ad un vicinato 5×5 ridotto; da un instruction set tipo CISC si passa ad un instruction set RISC; l'organizzazione logica della memoria immagine viene modificata così come la struttura del bus dei dati; viene inoltre integrata un'interfaccia con il sensore (telecamera) per l'acquisizione di immagini direttamente in memoria. Queste modifiche architetturali portano ad un incremento delle prestazioni di circa due ordini di grandezza.

Il sistema PAPRICA è stato principalmente impiegato nel settore automobilistico per l'elaborazione in tempo reale di immagini acquisite da una telecamera a bordo veicolo. Il risultato dell'elaborazione viene analizzato da processori di alto livello per inviare avvertimenti al guidatore. Gli algoritmi sviluppati sono finalizzati all'identificazione della corsia, della carreggiata, all'individuazione di eventuali ostacoli sul percorso, e alla determinazione dell'eventuale tempo di collisione con essi. Le prestazioni ottenute sono state incoraggianti perfino con la prima versione del prototipo, mentre le simulazioni del nuovo sistema PAPRICA-3 sono tali da far prevedere l'impiego di tale sistema (o di un sistema con architettura analoga) sui veicoli del futuro.

Appendice A

Speeding-up Mathematical Morphology Computations

In the literature a lot of different special-purpose architectures has been developed in order to exploit the special nature of the elaboration performed by morphological operators. A considerable part of them exploits spatial parallelism following the SIMD classification, in which a single pixel is elaborated by its own processing element. The main problem of this kind of architectures lays in the interconnecting topology, which limits the size of practical structuring elements. Some other architectures that have been investigated are formed by a number of specialized processors that operate in pipeline on pixel values. Furthermore, a lot of work has been devoted to the speed-up of morphological operations on general-purpose serial architectures, both using special data representations and performing efficient structuring elements decompositions.

The following section presents and discusses the implementation of mathematical morphology operations on three different kinds of architectures, showing the computing algorithms as well as their main performance indexes, together with some examples.

A.1 Hardware Platforms

In this section the application of mathematical morphology operators on three different kinds of architectures is investigated, starting from general-purpose serial architectures, covering pipeline systems, and finally presenting and discussing the philosophy of cellular processors. The efficiency of the algorithms used to compute morphological operations depends strongly on the particular hardware architecture; thus, for each considered system, a natural technique for computing a dilation is presented. Moreover, the performances depend on different parameters, which are analyzed and compared, showing some speed-up methods explained by some examples. Generally, both in the case when the use of a large structuring element is not efficient (on serial systems), or when it is impossible (due to hardware constraints in small-neighborhood based parallel systems), the solution to the problem of structuring element decomposition becomes of basic importance. The decomposition algorithms are based on different optimality criteria [82] in order to exploit the special hardware characteristics of each system. It is anyway important to remember that in the literature some other decomposition criteria, such as the decomposition of binary images for object recognition purposes [77], are discussed.

The following sections present some mathematical morphology computing techniques, based on different hardware platforms, and devise some speed-up methods mainly based on structuring elements decomposition.

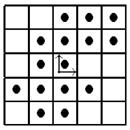
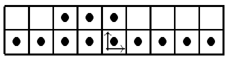
General-Purpose Serial Architectures

In general-purpose systems, when used to run morphological image analysis tasks, there is no upper bound to the size of structuring elements: in fact, using a bit-mapped representation of images, during the computation of the new value of an image pixel the processing element can access the values stored in every other image pixel within a constant time. On the other hand, the computational complexity of serial morphological operations depends on the number of elements which form the morphological operands. As an example, the computation of $A \oplus B$ requires one vector sum and one logical union for each couple of elements $a \in A$ and $b \in B$, and thus

$$\mathcal{C}(A \oplus B) = \#(A) \times \#(B) \ , \tag{A.1}$$

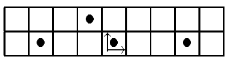
where $\mathcal{C}(\cdot)$ indicates the computational complexity of a given operation, and $\#(\cdot)$ represents the number of elements in a set. Some speed-up methodologies, based on different optimality criteria, have been developed for serial systems [82, 104, 39, 59, 103, 102]; they are based on the decomposition of convex structuring elements into a chain of dilations of simpler sets. The optimization criteria range from the minimization of the total number of computations (the one considered here), to the minimization of the number of decomposing sets, and to the minimization of the total number of elements in the decomposing sets. Although in [110] it has been proved that all convex structuring elements are algorithmically decomposable, unfortunately there is no answer to the problem of optimizing the automatic partitioning of non-convex sets into unions of convex ones for further decomposition.

Using the well-known visual representation of morphological sets, in dilation $R = I \oplus B$, where

$I =$  and $B =$  , (A.2)

the number of vector sums and logical unions, according to equation (A.1), is given by $\#(I) \times \#(B) = 15 \times 12 = 180$.

Using the following decomposition, the structuring element B can be expressed as the dilation between subsets B_1 and B_2:

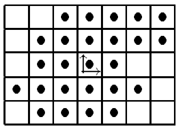
$B = B_1 \oplus B_2 =$  $\oplus$  ; (A.3)

using the *chain rule* property [61],

$$\begin{aligned} R &= I \oplus B = I \oplus (B_1 \oplus B_2) = \\ &= (I \oplus B_1) \oplus B_2 = R' \oplus B_2 \ , \end{aligned} \tag{A.4}$$

where $R' \triangleq I \oplus B_1$. The number of sums required to compute

$$R' = I \oplus B_1 = \tag{A.5}$$



is given by $\#(I) \times \#(B_1) = 15 \times 3 = 45$, while the number of sums required to complete the elaboration ($R = R' \oplus B_2$) is given by $\#(R') \times \#(B_2) = 25 \times 4 = 100$. Thus, the decomposition shown in the example, while requiring two dilations instead of one, decreases the total number of computations from 180 to 145.

The decomposition $B = B_1 \oplus B_2$ in order to offer computational savings when applied to $I \oplus B$ must obviously lead to the following relation:

$$\begin{aligned} \mathcal{C}(I \oplus B) \quad > \quad & \mathcal{C}\Big((I \oplus B_1) \oplus B_2\Big) = \\ & = \mathcal{C}(I \oplus B_1) + \mathcal{C}(R' \oplus B_2)\ . \end{aligned} \tag{A.6}$$

Considering the *overlapping* $\mathcal{O}(\cdot,\cdot)$ between two subsets A and B as a function $E^2 \times E^2 \mapsto Z$, defined as the number of equal elements in the set

$$C = \bigcup_{i=1}^{\#(A)} \bigcup_{j=1}^{\#(B)} \Big(a_i + b_j\Big)\ , \quad \text{with } a_i \in A, b_j \in B\ , \tag{A.7}$$

namely

$$\mathcal{O}(A, B) = \#(A) \times \#(B) - \#(A \oplus B)\ , \tag{A.8}$$

after few algebraic manipulations, the first part of relation (A.6) becomes:

$$\frac{\mathcal{O}(I, B_1)}{\#(I)} > \frac{\#(B_1)}{\#(B_2)} + \frac{\mathcal{O}(B_1, B_2)}{\#(B_2)}\ , \tag{A.9}$$

and, using expression (A.8),

$$\frac{\#(B) - \#(B_1)}{\#(B_2)} > \frac{\#(R')}{\#(I)}\ . \tag{A.10}$$

Equations (A.9) and (A.10) show that a decomposition is as much efficient (i.e. decreases the total number of computations) as the increment in the number of elements between the original image I and the set obtained by the first dilation R' is small; thus B_1 should be composed by a minimum number of elements with respect to B_2. Moreover, the overlapping between B_1 and B_2 should be kept small, and preferably should be zero.

On the other hand, the *logarithmic decomposition* [39, 111, 102] is another efficient method to speed-up morphological operations on large convex structuring elements. Such a decomposition minimizes the total number of points in the chain of decomposed sets, but it has the disadvantage to be of use only for convex structuring elements.

A generic non-convex structuring element B, at least in theory, can be decomposed as

$$B = B_1 \cup B_2 \cup ... \cup B_m \ , \tag{A.11}$$

where each B_i, $i = 1, ..., m$, is a convex set which can be further decomposed as shown before. Thus the computational complexity of a generic dilation $I \oplus B$ is given by:

$$\begin{aligned} \mathcal{C}(I \oplus B) &= \mathcal{C}\Big[(I \oplus B_1) \cup (I \oplus B_2) \cup ... \cup (I \oplus B_m)\Big] \\ &= \sum_{i=1}^{m} \mathcal{C}(I \oplus B_i) \ . \end{aligned} \tag{A.12}$$

In general-purpose serial architectures, the word parallelism range from 16 to 64 bits; when employed for binary morphological operations, this data parallelism is not fully exploited. In fact, using a $2^n \times 2^m$ pixels image, each vector sum involves only $n + m + 2$ bits[1], while each logical union operates on a single bit. Some special data packaging [104] can be used in order to operate in parallel on multiple image pixels, processing simultaneously p adjacent binary pixel values, allocated on the same word of parallelism $p \cdot (m + n + 2)$. With this solution the computational complexity still depends on the number of items in the structuring element, but now it is inversely proportional to the hardware word parallelism.

[1]Each element has two coordinates, m and n bits long; considering also an eventual carry for each operation, $n + m + 2$ bits are needed.

Some further application-dependent speed-up can be achieved when dealing with small decomposed structuring elements (such as 3×3 square-shaped or 8-connected neighborhood, and 3×3 diamond-shaped or 4-connected neighborhood). In fact, when a dilation is iterated (for example in *seed propagation*), only the pixels belonging to the external contour of binary objects are involved: the list of these pixels can be obtained easily from the previous iteration, increasing the overall performances by applying the dilation only to border pixels (*contour-processing* [105]).

Parallel Architectures

Two different parallel systems are considered in the following: architectures which distribute a single elaboration among different Processing Elements (PEs), implementing *operational parallelism*, and architectures which subdivide the processing of the whole data structure among different PEs running simultaneously the same code, implementing *spatial parallelism.* The next sections deal with systems involving these two different exploitations of parallelism: *pipeline* and *cellular* architectures.

Systems with Operational Parallelism

In *pipeline* systems, the PEs are generally specialized and thus devoted to different processings. Each PE is in charge of a different elaboration stage, sending the results to the next PE for further processing.

A lot of pipeline implementations have been discussed in the literature: the TAS [68], capable of performing three stages of binary morphological operations using an hexagonal structuring element; the Cytocomputer [83], which provides a serial pipeline of stages, applying a single 3×3 morphological or 8-bit algebraic operation on the whole image; the MITE [67], based on multiple pipelines with reconfigurable interconnections; the MVI Genesis 4000 [91], which consists of several different specialized stages (arithmetical, geometrical, and statistical), reaching a 10 Mbytes/s throughput.

In the following, the computational complexity of a morphological dilation is derived for pipeline systems formed by a set of PEs, each one operating on a single image pixel, and with the possibility to access a restricted neighborhood. This is a natural implementation of low-level image processing tasks [72], where image data come (generally in a raster-scan format [1]) from the acquisition system, and enter the first processing stage. This solution is particularly efficient when dealing with real-time constraints, since it allows to reach a considerably high throughput.

The method used to compute a dilation is based on the following property [61],

$$A \oplus B = \bigcup_{b \in B} (A)_b \ , \tag{A.13}$$

which shows a different technique, based only on operations easily implementable on pipeline systems, such as unions and translations. In fact, if the image data are loaded one at a time into a device that behaves like a shift register, a translation of the original image is just a delayed output of the shift register. Moreover, some additional hardware may be needed to prevent wrap-around of border pixels, as indicated in [1]. Unfortunately, delay elements can provide only translations in the direction opposite to the scanning direction (generally in row-major order, according to the axis orientation in figure A.1.a). Thus

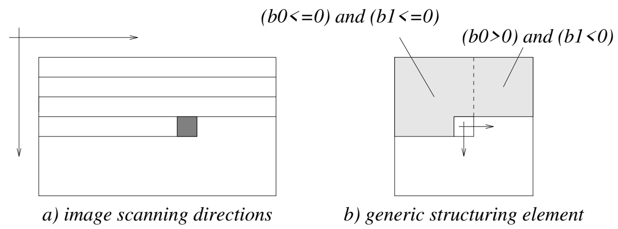


Figura A.1. Data scanning in pipeline systems

a structuring element B, containing at least one item $b = (b_0, b_1)$ such that $\big((b_0 \leq 0) \cap (b_1 \leq 0)\big) \cup \big((b_0 > 0) \cap (b_1 < 0)\big)$, as indicated in figure A.1.b, when used to parametrize a dilation, implies a non-causal delay, which cannot be performed. Recalling the translation invariance property of dilation [61]

$$(A)_x \oplus B = (A \oplus B)_x \ , \tag{A.14}$$

if $\tilde{b}$ represents the most astringent non-causal delay, the problem can be solved performing an initial causal delay $(-\tilde{b})$ on the input image A. Thus dilation $(A)_{-\tilde{b}} \oplus B$ contains only causal delays. The final result

$$\begin{aligned} (A)_{-\tilde{b}} \oplus B &= (A \oplus B)_{-\tilde{b}} = \\ &= \Big(\bigcup_{b \in B} (A)_b \Big)_{-\tilde{b}} = \bigcup_{b \in B} (A)_{b-\tilde{b}} \end{aligned} \tag{A.15}$$

can now be recovered by just a translation $(\tilde{b})$ in the output buffer.

The most important constraint which limits the dimensions of practical structuring elements is the high number of delay elements required by morphological operations on large structuring elements. Thus, in this case a decomposition is required to reduce the hardware complexity of the system; as a drawback, it introduces some extra processing steps, which anyway can be assigned to different PEs.

Let's consider a pipeline in which all the elaborations are synchronized. If in a generic dilation $A \oplus B$ the structuring element B can be decomposed in $B = \Big(\Big((B_1 \oplus B_2) \oplus B_3 \Big) \ \oplus \ \dots \ \Big) \oplus B_n$, where each B_i has a dimension smaller than the neighborhood accessible by each PE, and if the number of PEs is larger than n, then the chain of dilations can be distributed among all the PEs. Each PE iterate its elaboration on every image pixel a_i, as soon as the data is available to the first pipeline stage. Since each dilation is performed in parallel by the chain of PEs, in stationary conditions the computational complexity is given by:

$$\mathcal{C}(A \oplus B) = \mathcal{F}_x \times \mathcal{F}_y \times \mathcal{C}(a_i \oplus B_j) \ , \tag{A.16}$$

where $\mathcal{F}_x$ and $\mathcal{F}_y$ represent the frame dimensions, and $\mathcal{C}(a_i \oplus B_j)$ indicates the complexity of a dilation between an image composed by a single element a_i and a generic structuring element of the set $\{B_1, B_2, ..., B_n\}$; the latter represents the complexity of the dilation associated to any single pipeline stage. Thus, for sufficiently long pipeline systems, the complexity of a generic morphological operation based on a convex structuring element is independent of the structuring element dimensions, while it depends directly on the image dimensions.

Equation (A.12), introduced in the case of serial architectures, indicates also the computational complexity of morphological operations based on non-convex structuring elements performed on this kind of pipeline systems. Thus, if the system is composed by multiple pipelines operating in parallel, such as the MITE [67], the total computational complexity can be reduced assigning each sub-elaboration of convex sets ($I \oplus B_i$, according to equation (A.12)) to a different pipeline.

Systems with Spatial Parallelism

When a bit-mapped data representation is used, mathematical morphology operations involve repeated computations over large data structures, thus any form of spatial parallelism improves the overall performances. Parallel architectures with spatial parallelism are formed by a high number of PEs devoted to the simultaneous elaboration of a single image area.

A number of processor arrays with a bidimensional grid interconnection scheme and a SIMD processing paradigm have been conceived, designed and implemented, starting from the original ideas of Unger [101]. Fountain [49] has classified in chronological order the different research projects and implementations according to a three generation taxonomy. The early ILLIAC [13] and CLIP [45] machines belong to the first generation, mainly devoted to low level image processing tasks. The second generation comprises systems such as the ICL DAP [81], the Goodyear MPP [9] and the CLIP4 system [50] which have evolved to complete processing systems with dedicated operating systems and languages, and extended the application spectrum to high speed scientific computations.

The triggering factors of the third generation were the availability and widespread use of VLSI technologies of the 80's and the natural mapping of a bidimensional interconnection scheme over the planar structure of a silicon chip. This led to an explosion of different proposals and implementations [100, 74, 63, 62, 87, 55] and in some cases to the redesign of previous architectures. Most designs share original characteristics such as the bidimensional mesh interconnection scheme and bit serial computation while other ones, such as the Connection Machine [62], have increased the complexity of the interconnection network or widened the data path of the elementary PE, as in the CLIP7 [51] or MasPar [70] systems. A recent commercially available machine, the AIS [87], has a 1-D interconnection scheme emulating, via a specialized memory interface, a 2-D mesh organization.

In order to run mathematical morphology applications, in massively parallel architectures each pixel is assigned to a PE, which computes its new value, depending on the neighborhood configuration, which reflects the particular structuring element. The computational paradigm is then based on Cellular Automata [36, 46] and all the possible operations that the PEs can execute form the *instruction set*. Generally the interconnecting topology limits the dimensions of practical structuring elements, typically reducing it to 3×3 [47], as in the Cytocomputer and MPP implementations. Thus a generic structuring element needs to be decomposed into a chain of simpler and smaller ones, with the constraint of belonging to the instruction set.

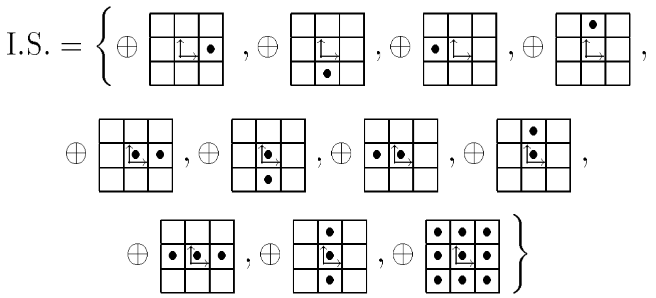
Generally in massively parallel architectures the integration of a high number of simple PEs is preferred with respect to a higher complexity of each single PE. The simple single-bit architecture of each PE allows to reach a hardware efficiency higher than the one obtained by architectures with a higher word parallelism, since the computations performed do not always involve multiple-bit operations. On the contrary, some Cellular Automata based architectures, such as the CAM-6 [98] or the CAM-8 [69], process from 4-bit data up to 16-bit data in parallel, raising the problem of grey-level structuring element decomposition [89, 111].

Thus, the efficiency of the computation depends on the instruction set, which must be accurately tuned. Next section leads to the definition of an algorithm for the optimization of the decomposition of generic binary structuring elements on a special-purpose cellular architecture, and it addresses the problem of the definition of an efficient instruction set.

A.2 Morphological Processing on Cellular Architectures

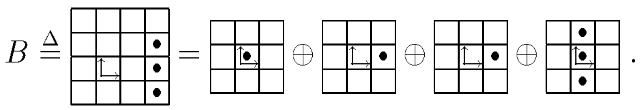
From the previous considerations, it is clear that the Array Processor is the architectural implementation that exploits in a better way the features of morphological operations.

When dealing with this kind of systems, a set is said to be *convex with respect to a specific instruction set* when it can be obtained starting from a single pixel overlapped onto the Origin, and applying only operations belonging to the instruction set of the specific system. Thus, if the structuring element is convex, then its decomposition can be easily devised by the convexity definition itself. As an example, using a cellular architecture with an instruction set composed by the following operations,



(A.17)

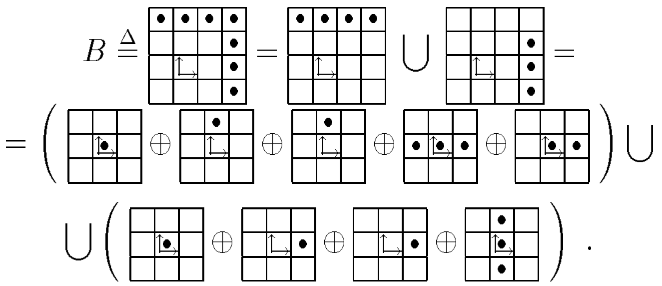
the following decomposition can be devised:



(A.18)

This example shows the synthesis of B starting from a single element in the Origin, and using only morphological operations belonging to the instruction set. Obviously, in order to enhance the probability that a generic structuring element is convex with respect to a specific instruction set, the instruction set itself should contain the higher number of instructions, and thus a first rough consideration is that a CISC-oriented implementation should be followed.

In the case when the generic structuring element B is non-convex, it must be subdivided according to equation (A.11). As already mentioned, the decomposition problem is equivalent to the synthesis of the structuring element starting from one (in the case of convex) or more than one (in the case of non-convex) elements on the Origin. An example of synthesis of non-convex set is given in the following:

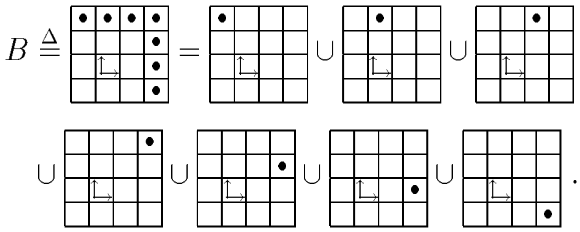


(A.19)

Thus, starting from a single element on the Origin, and applying only operations belonging to the instruction set, is there a rule determining the computational complexity for the synthesis of a generic $n \times m$ non-convex structuring element? In the following sections some algorithms are presented and their computational complexity computed in order to devise an efficient method for the automatic decomposition of generic structuring elements on the special-purpose cellular architecture PAPRICA.

Algorithms for Set Synthesis and their Complexity

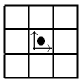
A rough indication of the computational complexity can be devised by considering a generic structuring element B as composed by the logical union of a number of convex sets equal to the number of elements in set B, each one formed by just one item b_i. For example:



(A.20)

In this case the synthesis of a generic single-element set, starting from the single item overlapped onto the Origin, is performed by a chain of simple translations. The complexity of this solution is then expressed by:

$$\mathcal{C}(B) = \sum_{i=1}^{\#(B)} \Big(\mathrm{d}(b_i, O)\Big) \ , \tag{A.21}$$

where the improper expression $\mathcal{C}(B)$, that will be used hereinafter, stands for $\mathcal{C}\Big($  $\oplus B\Big)$, and $\mathrm{d}(b_i, O)$ indicates the distance of element $b_i \in B$ from the Origin O using the metric of the specific instruction set[2].

Unfortunately this solution implies to execute the same translations more than once, decreasing the overall performances. A simple improvement to this algorithm consists in translating the initial element over the whole area defined by the structuring element, and keeping (with a union operation) only the intermediate results that correspond to any b_i, with $i = 1, 2, ..., \#(B)$. Thus, in this case:

$$\mathcal{C}(B) = m \times n \ , \tag{A.22}$$

where m and n are the dimensions of the minimum bounding box which contains both the structuring element and the Origin, as shown in figure A.2.

[2]If the instruction set (as the one presented in (A.17)) comprehends only single-step translations in the 4 main directions, then a diagonal translation implies 2 operations, and thus a 4-neighbors metric is used. The 8-neighbors metric is used when the instruction set comprehends also diagonal translations. In the following, only the 4-neighbors metric will be used.

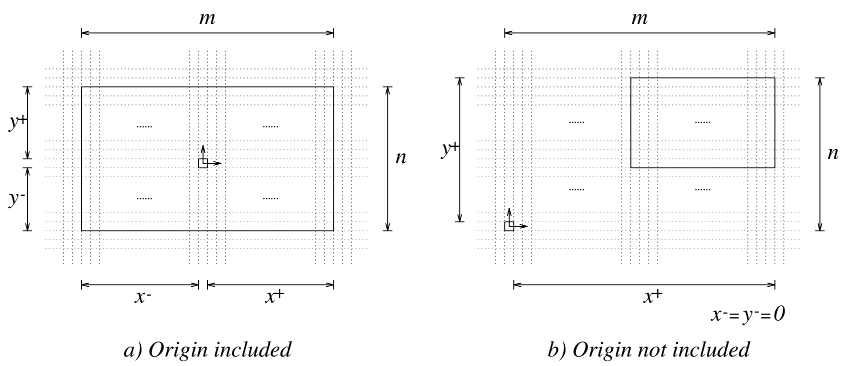


Figura A.2. Decomposition parameters

It is clear that in the previous solution some translations are useless. This consideration provides a minor improvement to the synthesis algorithm: the initial element could be translated only onto useful positions. Being $e = \#(B)$, the elements of B can be ordered in $e!$ different sequences S_j:

$$S_j = \left\{ b^{(j,1)} \rightarrow b^{(j,2)} \rightarrow \ldots \rightarrow b^{(j,e)} \right\} , \tag{A.23}$$

with $j = 1, 2, ..., e!$. Thus the computational complexity can be reduced to

$$\mathcal{C}(B) = \min_j \sum_{i=1}^{e} \mathrm{d}\left(b^{(j,i-1)}, b^{(j,i)} \right) , \tag{A.24}$$

where $b^{(j,0)}$ represents the Origin. Note that the determination of the minimum coincide with the solution of the traveling salesman problem. Since, if $O \notin B$, $\mathrm{d}(b^{(j,i-1)}, b^{(j,i)}) \geq 1$, a lower bound can be detected:

$$\mathcal{C}(B) \geq e = \#(B) , \tag{A.25}$$

which becomes

$$\mathcal{C}(B) \geq e - 1 = \#(B) - 1 , \tag{A.26}$$

in the case of $O \in B$.

Being x^+, x^-, y^+, and y^- the maximum distances (considered 0 if negative) between the elements of B and the Origin as shown in figure A.2, another lower bound can be found replacing them in equation (A.24):

$$\mathcal{C}(B) \geq x^+ + x^- + y^+ + y^- \ . \tag{A.27}$$

Since $x^+ + x^- = m - 1$ and $y^+ + y^- = n - 1$,

$$\mathcal{C}(B) \geq (m-1) + (n-1) \ . \tag{A.28}$$

Furthermore, exploiting the main feature of cellular systems, namely the capability to perform synchronously the same operation on more than one item, a better improvement can be achieved.

The RCT Algorithm

A new algorithm for parallel and cellular structuring element synthesis is presented in the following.

(a) The element overlapped onto the Origin is translated horizontally in both directions until both the ends of the bounding box of figure A.2 are reached, keeping each intermediate result. This step requires a total of m translations.

(b) Each row of the structuring element is composed with logical operations, using all the temporary sets obtained during step (a), and stored in different temporary sets (binary image planes).

(c) Finally, the rows are translated using a mechanism that reduces the number of translations to n.

This algorithm, based on Row/Column Translations (RCT algorithm) is presented in the following example, which shows the synthesis of set B of expression (A.19):

$$B \triangleq \begin{array}{|c|c|c|c|}\hline \bullet & \bullet & \bullet & \bullet \\ \hline & & & \bullet \\ \hline & & & \bullet \\ \hline & & & \bullet \\ \hline\end{array} \tag{A.29}$$

(a) Synthesis of temporary sets T_i:

$$\begin{aligned} T_1 &= \begin{array}{|c|c|c|}\hline & & \\ \hline & \bullet & \\ \hline & & \\ \hline\end{array} \oplus \begin{array}{|c|c|c|}\hline & & \\ \hline \bullet & & \\ \hline & & \\ \hline\end{array}\ ; \quad T_2 = \begin{array}{|c|c|c|}\hline & & \\ \hline & \bullet & \\ \hline & & \\ \hline\end{array}\ ; \\ T_3 &= \begin{array}{|c|c|c|}\hline & & \\ \hline & \bullet & \\ \hline & & \\ \hline\end{array} \oplus \begin{array}{|c|c|c|}\hline & & \\ \hline & & \bullet \\ \hline & & \\ \hline\end{array}\ ; \quad T_4 = T_3 \oplus \begin{array}{|c|c|c|}\hline & & \\ \hline & & \bullet \\ \hline & & \\ \hline\end{array} \end{aligned} \tag{A.30}$$

(b) Logical composition of rows R_i:

$$\begin{aligned} R_1 &= T_1 \cup T_2 \cup T_3 \cup T_4\ ; \quad & R_2 &= T_4\ ; \\ R_3 &= T_4\ ; & R_4 &= T_4 \end{aligned} \tag{A.31}$$

(c) Synthesis of structuring element B:

$$\begin{aligned} C_1 &= R_1\ ; & C_2 &= C_1 \oplus \begin{array}{|c|c|c|}\hline & \bullet & \\ \hline & & \\ \hline & & \\ \hline\end{array} \cup R_2; \\ C_3 &= C_2 \oplus \begin{array}{|c|c|c|}\hline & \bullet & \\ \hline & & \\ \hline & & \\ \hline\end{array} \cup R_3; & B &= R_4 \oplus \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & \bullet & \\ \hline\end{array} \cup C_3 \end{aligned} \tag{A.32}$$

A new formulation of the computational complexity, based on the RCT algorithm, is then presented: following the previous considerations, it is

$$\mathcal{C}_{\mathrm{RCT}}(B) = (m-1) + (n-1)\ . \tag{A.33}$$

The previous expression shows that the performances of the algorithm, with complexity $\mathrm{O}(n+m)$, are independent of the shape of the specific structuring element: this algorithm can in fact be used also to synthesize any generic non-convex structuring element. On the other hand, the synthesis of a generic structuring element B with a serial architecture is an $\mathrm{O}(e)$ problem, where $e = \#(B)$.

It is important to note that in the determination of the computational complexity in equation (A.33), only morphological operators based on a neighborhood larger than 1×1 are considered. Their execution imposes that each PE accesses the memory of other PEs, in order to fetch the processing data. On the other hand, the execution of 1×1 morphological operators, namely *logical* operations, requires only the access to the local memory of each PE, and thus they are performed more efficiently.

The RCT algorithm is based only on single-step translations in the 4 main directions, but which is the performance improvement when more general structuring elements can be used in addition to single-step translations and logical unions? The operations that a generic cellular architecture can typically implement in hardware are a subset of all the possible 3×3 structuring elements with the origin in the center $\left(\begin{array}{|c|c|c|}\hline & & \\ \hline & \llcorner & \\ \hline & & \\ \hline\end{array}\right)$, which may contain also the translations in the 4 main directions as special cases.

The following example shows how the RCT algorithm can be improved using the specific set of 3×3 structuring elements implemented on the special-purpose massively parallel architecture, PAPRICA [26, 55, 22], but can be easily extended to any other cellular architecture holding its specific instruction set.

Set Decomposition on a Special-Purpose Architecture

The following example shows a generic non-convex set. In this case the structuring element B is composed by a union of equal non-overlapped convex sets:

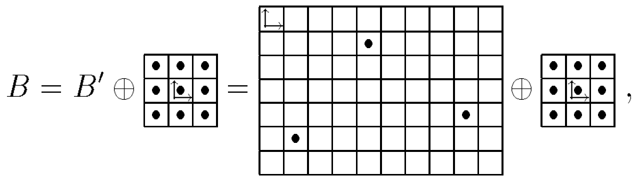
$$B \triangleq \begin{array}{|c|c|c|c|c|c|c|c|c|c|}\hline \llcorner & & & \bullet & \bullet & \bullet & & & & \\ \hline & & & \bullet & \bullet & \bullet & & & & \\ \hline & & & \bullet & \bullet & \bullet & & & & \\ \hline & & & & & & & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & & & & & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & & & & & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & & & & & & & \\ \hline\end{array} \tag{A.34}$$

The RCT algorithm would lead to the following computational complexity:

$$\mathcal{C}_{\mathrm{RCT}}(B) = (m-1) + (n-1) = 9 + 6 = 15 \ . \tag{A.35}$$

PAPRICA instruction set, already introduced in (A.17), is based on more general 3×3 structuring elements; this section shows a decomposition algorithm that exploits the special features of this special-purpose architecture, and reaches better performances with respect to the RCT algorithm. The following presentation focuses deliberately only on PAPRICA instruction set implementation, rather than the architectural details of the system.

For example, performing an initial 8-directions dilation,



(A.36)

the new set B' can now be decomposed following the RCT algorithm, achieving a lower computational complexity. In fact,

$$\mathcal{C}_{\mathrm{RCT}}(B') = 8 + 5 = 13 \ , \tag{A.37}$$

and thus, adding the complexity of the 8-directions dilation,

$$\mathcal{C}(B) = 1 + \mathcal{C}_{\mathrm{RCT}}(B') = 1 + 13 = 14 \ . \tag{A.38}$$

The advantage lays in the fact that with a single operation both the number of rows and columns in the resulting set have been reduced. Moreover, if the Origin was placed within the bounding box that contains the 3 elements, the complexity reduction would have been much more sensible (reaching $\mathcal{C}(B) = 1 + (7 + 4) = 12$).

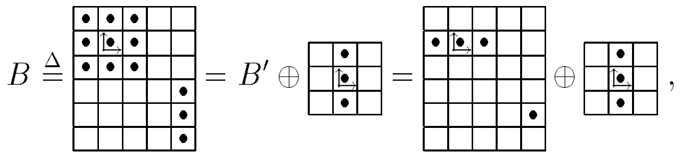
Thus, the use of more general dilations allows computational savings only when it removes more than one column and/or row external with respect to the Origin. As a consequence, the complexity based on the RCT algorithm is the lower possible unless the application of any other dilation reduces by an amount of 2 or more the dimensions of the resulting set.

In order to formalize the final formulation of the decomposition algorithm, let's distinguish between two different cases:

1. there exist at least one dilation that can be applied to the whole set, as in example (A.36), decreasing by an amount of 2 or more the dimensions of the resulting set;

2. there exist a dilation which can be applied only to a subset of the original structuring element, decreasing the dimensions of one of the resulting sets.

An example of the first case is given in the following:

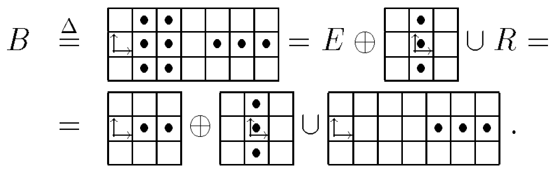


(A.39)

where $\mathcal{C}_{\mathrm{RCT}}(B) = 4 + 5 = 9$, while applying a vertical dilation $\mathcal{C}(B) = 1 + \mathcal{C}_{\mathrm{RCT}}(B') = 1 + (4 + 3) = 8$.

If a dilation reduces only by 1 the sum of the dimensions of the set, then its application does not modify the total computational complexity, since the other modifications that it could eventually produce are internal with respect to the bounding box of the resulting set, which normally fixes the complexity of the synthesis of that set, as shown in [28].

On the other hand, an example of the second case is given in the following: it shows the partial decomposition of a set B into two subsets: the eroded one (E) and the remainder one (R). In this case a logical union is required:



(A.40)

In this case the computational complexity, beside the dilation operation, must take into account the decomposition of both the eroded and remainder sets;

thus $\mathcal{C}(B) = 1 + \mathcal{C}(E) + \mathcal{C}(R) = 1 + 2 + 6 = 9$, while $\mathcal{C}_{\mathrm{RCT}}(B) = 6 + 2 = 8$. The increment in the computational complexity is due to the fact that, on single-bit PEs, each operation (in this case a translation) is performed on a single binary set; thus, if the synthesis of the two sets E and R requires the same operations, they will be executed twice. Such a decomposition may be useful when the first part of the synthesis of E and R involve the same elaborations, that can thus be executed only once, as shown in figure A.3 and in the following example.

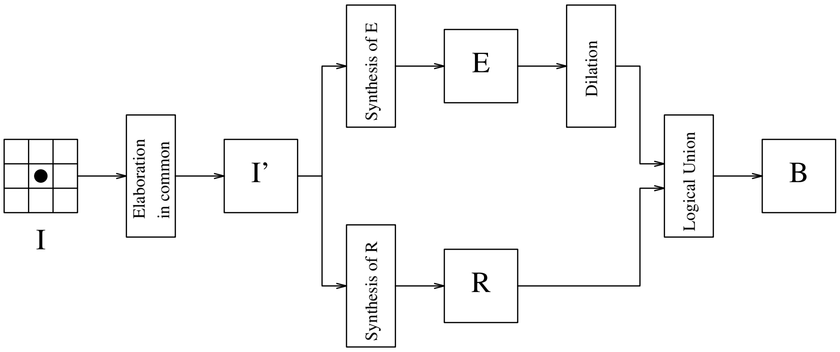
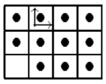
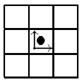
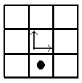
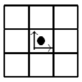
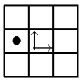
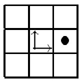
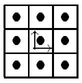


Figura A.3. Processing in the case of decomposition with remainder

The synthesis of set $B \triangleq$  can be obtained using one temporary set $I' =$  $\oplus$  :

$B = \Big[\Big($  $\cup I'\Big) \oplus$  $\Big] \bigcup$

$\bigcup \Big[\Big(I' \oplus$  $\Big) \oplus$  $\Big]$ (A.41)

Thus, following this last formulation of the algorithm, it is $\mathcal{C}(B) = 4$, while $\mathcal{C}_{\mathrm{RCT}}(B) = 5$.

The only rule missing in the presented algorithm refers to the problem of choosing, in case of possibility, which one of the two previous cases leads to the best decomposition. The following example shows a set B that admits two different decompositions following the two previous cases (note that $\mathcal{C}_{\mathrm{RCT}}(B) = 13$):

$$B \triangleq \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & \bullet & \bullet & \bullet \\ \hline & & & \bullet & & \bullet & \bullet & \bullet \\ \hline & & & \bullet & & \bullet & \bullet & \bullet \\ \hline & & & \bullet & & & & \\ \hline \bullet & \bullet & \bullet & & & & & \\ \hline \bullet & \bullet & \bullet & & & & & \\ \hline \bullet & \bullet & \bullet & & & & & \\ \hline \end{array}\,. \tag{A.42}$$

Decomposition following case 1: $\quad (\mathcal{C}(B) = 12)$

$$B = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & & & \\ \hline & & & & & \bullet & \bullet & \bullet \\ \hline & & & \bullet & & & & \\ \hline & & & & & & & \\ \hline & & & & & & & \\ \hline \bullet & \bullet & \bullet & & & & & \\ \hline & & & & & & & \\ \hline \end{array} \oplus \begin{array}{|c|c|c|}\hline & \bullet & \\ \hline & \bullet & \\ \hline & \bullet & \\ \hline \end{array}\,. \tag{A.43}$$

Decomposition following case 2: $\quad (\mathcal{C}(B) = 11)$

$$B = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & & & \\ \hline & & & & & & \bullet & \\ \hline & & & & & & & \\ \hline & & & & & & & \\ \hline & & & & & & & \\ \hline & \bullet & & & & & & \\ \hline & & & & & & & \\ \hline \end{array} \oplus \begin{array}{|c|c|c|}\hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \end{array} \bigcup \begin{array}{|c|c|c|}\hline & \bullet & \\ \hline & \bullet & \\ \hline & \bullet & \\ \hline \end{array}\,. \tag{A.44}$$

Note that in the specific case of PAPRICA architecture, expression (A.43) could be further improved, thanks to the presence of a horizontal dilation.

The final formulation of the algorithm for set decomposition on architectures with the possibility to perform a fixed set of 3×3 dilations and only logical unions is thus summarized in the following.

1. If the application of a dilation that can lead to an eroded set whose dimensions sum is smaller by an amount of 2 or more with respect to the original one B does not exist in the instruction set, then proceed with the RCT algorithm;

2. otherwise, one or both of the two following cases are possible:

   (a) $B = E_1 \oplus D_1$, and/or

(b) $B = E_2 \oplus D_2 \cup R$,

where D_1 and D_2 represent the dilations that achieve the best dimensions reduction for each case. Note that there is no penalty in this choice: in fact, due to the chain rule property [61], if a dilation would produce better results, it could be swapped and become the first dilation to be applied.

- If only one case is possible, then proceed with that decomposition;
- otherwise, if $\mathcal{C}(E_1) \leq \mathcal{C}(E_2)$, then proceed with decomposition (a);
- otherwise, decomposition (b) should be chosen only if there exist an intermediate set I' such that (following notations of figure A.3)

$$\begin{aligned} \mathcal{C}(E_1) + 1 > \mathcal{C}(I') + \mathcal{C}(I' \to E_2) + \\ + \mathcal{C}(I' \to R) + 1 \ , \end{aligned} \tag{A.45}$$

where $\mathcal{C}(A \to B)$ indicates the computational complexity of the synthesis of set B starting from set A; note that, in this specific case, $\mathcal{C}(A \to B) = \mathcal{C}(B) - \mathcal{C}(A)$. Thus, from expression (A.45), if

$$\mathcal{C}(E_1) > \mathcal{C}(E_2) + \mathcal{C}(R) - \mathcal{C}(I') \ , \tag{A.46}$$

then proceed following case (b).

It must be noted that in order to choose between case (a) and (b), both decompositions must be investigated; the decomposition was thus implemented through a recursive algorithm.

Appendice B

Focus of Attention Emulation on PAPRICA System

B.1 Problem Definition

Generally the processing has a different relevance for different image areas. This fact allows to decrease the number of computations, skipping the processing of the image regions which won't produce significative results, and implementing a sort of a multiple "focus of attention" [109, 75].

The main problem is anyway to detect the different image areas where to perform or skip the elaboration. Generally, in vision systems the way to implement the "focus of attention" mechanism is demanded to the medium-level stages of processing [5]: i.e. the low-level system performs general filterings (early preattentive parallel processing); the medium-level extracts the significant features, detecting the areas of interest (cognitive serial processing: attention-driven visual search), and drives the next low-level processing, using a feedbacked model. The main disadvantage of this approach lays in the modification of the data structure, which changes from a 2D image, to a higher level description of areas, and back to a 2D image again. For this reason this approach does not fully exploit the SIMD parallelism of a massively parallel architecture, requiring a different computing platform.

The "focus of attention" can also be implemented remaining within the same 2D data structure, using a single SIMD architecture [60]; in this case, the speed-up is obviously achieved only if the number of PEs is less than the number of pixels, and in particular, when a specific PE virtualization mechanism is used. For example, the Connection Machine CM-2 [62, 94] virtualization mechanism is not suited for this kind of speed-up, since the conditional elaborations are performed using a context bit for each PE. More precisely, several data (belonging to different pixels) are loaded into each PE, and the computation, serialized within each PE, is performed anyway, discarding the results associated to the pixels whose context bit is reset. This virtualization mechanism does not lead to a reduction in the number of computations.

In the last few years, a lot of retina-like sensors (such as the one discussed in [96]), with a high resolution array in the central area (fovea) and a coarse one in the remainder, have been designed and implemented, imposing a hardware constraint on the position of the focus of attention. Nowadays, taking advantage of the technology evolution, it is possible to integrate high resolution sensors with arrays of 1024×1024 pixels [86] or, in the near future, even more; the high resolution distributed over the whole sensor offers the advantage to move the position of the focus of attention with software algorithms, according to real-time inputs.

B.2 An Architectural Enhancement for Focus of Attention Emulation on PAPRICA

The efficient implementation of the "focus of attention" is possible thanks to PAPRICA PE virtualization mechanism. The triggering idea is based on the consideration that sometimes, after the loading of an image sub-window into the PA, the elaboration may be useless, and thus the operation of storing back the results into the image memory wouldn't be required. The choice whether to perform the elaboration or not depends on the characteristics of the complete set of data loaded into the PA internal registers (Q^2 elements). Each 16-bit element passes through the 16-bit bus that links the image memory to the PA, and it is captured by the additional circuit shown in figure B.1.

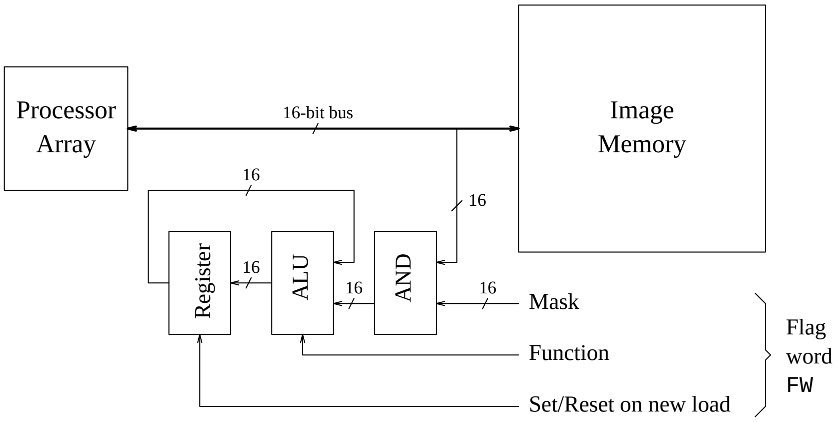


Figura B.1. Hardware modification for the implementation of the "focus of attention"

Only a set of bits of the captured values are kept, thanks to a logical intersection operation (*AND*) with a programmable mask; then the result and the value contained into the register are sent into an ALU. The operation performed by the ALU can be selected within the following set: {*OR*, *AND*, *XOR*, *NOR*, *NAND*, *EQU*, *MAX*, *MIN*}, while the result is stored back again into the register. When a new image sub-window is loaded into the PA, the register is set or reset according to a specific programming input.

Finally, the elaboration and the consequent storing of the results can be conditioned to a particular bit (or set of bits) of the register. In the following example, written in PAPRICA assembly language, the elaboration is skipped when all the pixels of the first PA layer are set to 0.

```
SET_FW FW_OR FF_RESET BLOCK0 0x000f OB_RESET
               ; Set the mask to -1, the ALU function
               ;to OR, and RESET the register on new
               ;sub-window loading
IFFW L$1       ; If the PA contains at least 1 bit
               ;set in the 1st layer, then
  ...          ; perform the elaboration
ELSEFW         ; otherwise
  NO_WRITEBACK ; don't perform anything and skip the
               ;write-back procedure
ENDIFFW
```

The processing obtained in this case is substantially pyramidal: a single value is computed for each image sub-window loaded into the PA; the set of these values forms a coarse resolution mask. Then, the particular value associated to each sub-window selects the different fine-grain processing that will be performed by the PA.

An important consideration is that the resolution of the coarse-grain mask depends on the PA dimension. A small PA causes the mask to have a small granularity, allowing a fine distinction between the different areas, but reduces the parallelism in the computation (small number of PEs), and thus the processing speed. A large PA improves the performances in terms of speed (high number of PEs), but it has the disadvantage to characterize coarsely the different regions.

The previous example can be generalized: it is frequent in vision applications that a binary image-mask is used to detect two different kinds of regions in order to perform two different elaborations (e.g. smoothing vs enhancement). As a comparison, the implementation of this kind of different processing (*data-driven focus of attention*) on a CM-2 is less efficient, since both the computations are performed on every pixel (complementing the context bit), keeping only the significant results.

The extension of PAPRICA controller can be useful both to speed-up the elaboration, processing only the relevant areas of the image or implementing different elaborations with a high efficiency, and to implement functions that otherwise couldn't be implemented, such as the global computation of the maximum value in a n-bit image. Obviously the performance improvement depends on the processed image, but it is important to note that there is no overhead in the worst case (namely when each sub-window elaboration is required) with respect to the present solution.

B.3 A Few Examples

The two case studies that are presented in the following sections are: a thinning filter iterated on a binary image, and the computation of the maximum value in a 256×256 8-bit image. The following values are assumed: PA dimension $Q^2 = 16^2$; memory cycle time $T_M = 250$ ns; array cycle time $T_C = 500$ ns, reflecting the current implementation of PAPRICA architecture.

The performances of the two presented applications have been derived with the help of a software simulator implementing the proposed extension, and will be compared to the ones obtained with the existent PAPRICA hardware board.

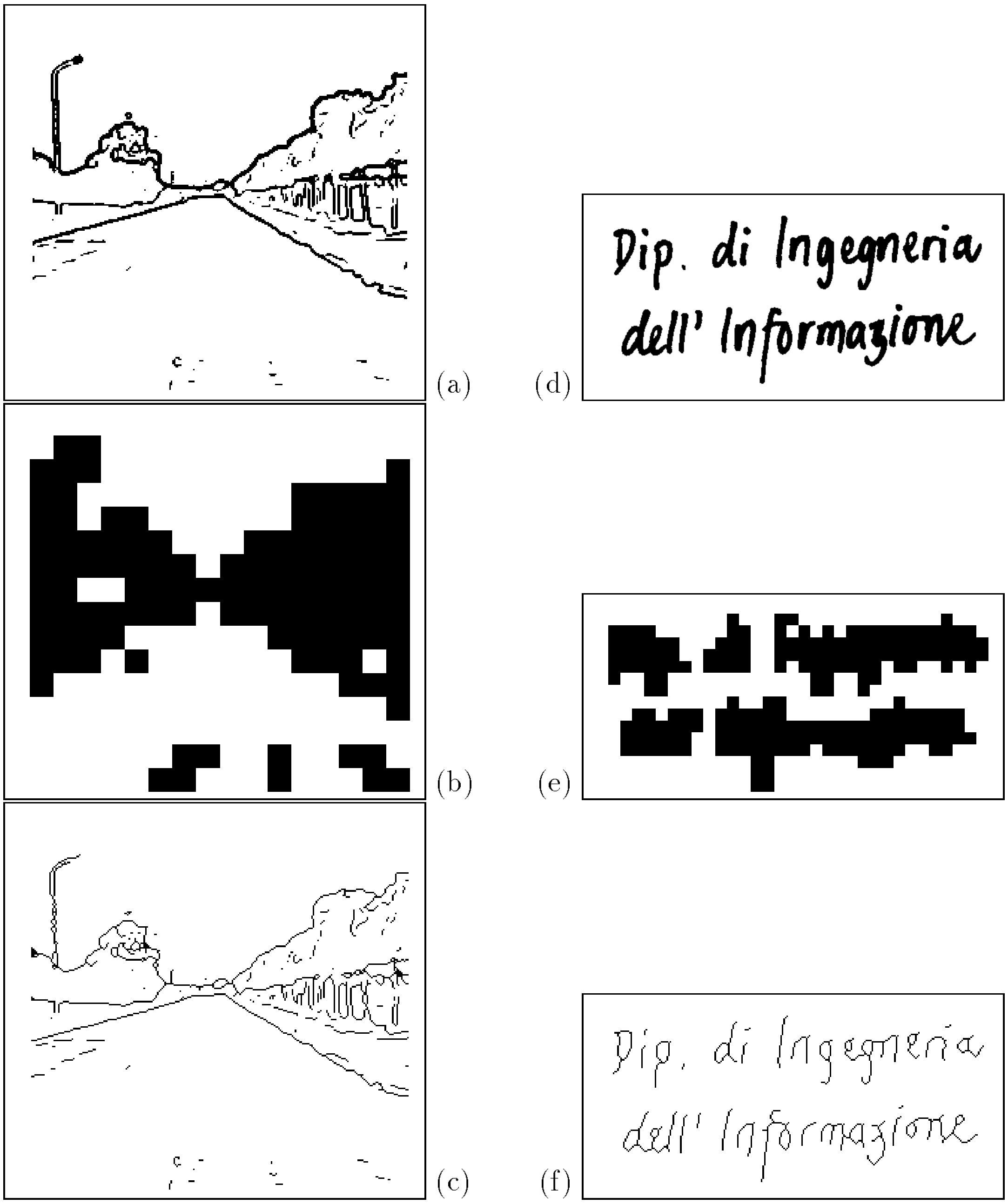


Figura B.2. Original, area of interest, and thinned images: *a*) original "campus" image (256×256 pixels); *b*) area of interest; *c*) thinned image (3 iterations); *d*) original "address" image (512×256 pixels); *e*) area of interest; *f*) thinned image (5 iterations).

A Binary Morphology Transform: Thinning

The thinning transform enables only the pixel state transitions from the *foreground state* to the *background state*, which maintains the original connectivity of both the objects and the background. The final result, obtained after a finite number of iterations, depending on the particular image, appears as the skeleton of the image objects; all the pixels of the skeleton have only two neighbors, except the branch points, which have more than two, and the end points, which have only one neighbor.

In [84] a wide class of parallel thinning algorithms is presented; the algorithm used here is based on the one presented in [7] where the correctness of the algorithm is proven. It reduces the thinning algorithm to an iterative *composite matching*, which is the natural PAPRICA computational paradigm [26, 56].

In this case the speed-up is based on the consideration that if the sub-window loaded into the PA is completely set or reset, then the computation is useless, since it would not modify the values stored into the PA. Thus, in this case the whole elaboration is skipped, as well as the rewriting of the result back into the image memory.

	iter. #1	iter. #2	iter. #3
m	582048	581760	581760
c	16117	16097	16097
w	906	904	904
t [ms]	154	153	153
s [Mpixel/s]	0.42	0.43	0.43
speed-up %	31.25 %	34.38 %	34.38 %

Tabella B.1. Performances of the thinning filter on “campus” image (256×256)

Tables B.1 and B.2 show the performances of the first iterations of the thinning filter (referred respectively to the two images in figure B.2, and computed with the current PA dimension $Q^2 = 256$) in terms of memory accesses (m), PA clock cycles (c), windows processed (w), computational time (t), processing speed (s), and speed-up percentage with respect to the current hardware implementation of PAPRICA.

	iter.#1	iter.#2	iter.#3	iter.#4
m ($\times 10^6$)	1.205	1.193	1.179	1.172
c	35249	34379	33449	32959
w	2080	1993	1900	1851
t [ms]	319	315	312	310
s [Mp/s]	0.41	0.42	0.42	0.42
speed-up	28.12%	30.00%	31.25%	32.18%

Tabella B.2. Performances of the thinning filter on "address" image (512×256)

Table B.3 shows the dependence of the performances from the PA dimension Q^2, considering only the first thinning iteration on "campus" image and the same frame dimension (256×256).

Q^2	16×16	32×32	48×48	64×64
speed-up	31%	19%	11%	4%

Tabella B.3. Performances dependency from the PA dimension Q^2

The PAPRICA implementation of the thinning filter is a typical I/O bounded algorithm [22, 26], where the computational time is negligible with respect to the data transfer time. Therefore in this case the speed-up is due almost exclusively to the elimination of some data transfers, producing a maximum of a

50% increment in the final processing speed. However, the speed-up capability is mainly exploited with CPU-bounded algorithms, where the elimination of a few sub-window processings turns into a sensible decrement also in the number of PA clock cycles.

Currently the system simulation is used also to compare the previous results to the performance improvement achieved in the case of optical flow computation (a typical CPU-bounded algorithm), performed only on the significant image areas, showing a more sensible speed increment.

Global Function Evaluation

The main goal that led to the design of the new controller is the possibility to skip useless computations and data transfers, as shown in the previous paragraph. Nevertheless, a useful side effect is that it allows the computation of global functions on the whole image (e.g. the maximum value) without any further increment in the controller complexity.

The example of maximum value evaluation is shown in the following. During the loading of the values of each sub-window into the PA, the local maximum is computed in the new register; it is then copied into each PE and the result is stored back into the image memory. The number of memory accesses is $2\ 256^2$, namely two complete transfers of the whole image. Then, using PAPRICA pyramidal capability, the linear resolution is reduced of a factor 16, fetching 1 pixel from each previous sub-window, and finally computing the global maximum, which will be stored back into the same pixels. The number of memory accesses is $2\ 16^2$, namely two complete transfers of a single sub-window. The performances are summarized in table B.4.

If some *ad hoc* hardware would be implemented for the determination of the

	64×64	128×128	256×256	512×512
m	8512	33152	131584	526848
c	435	1635	6435	25776
w	17	65	257	1029
t [ms]	2.35	9.1	36.1	144
s [Mp/s]	1.74	1.80	1.81	1.81

Tabella B.4. Performances of the maximum evaluation in a 8-bit image

maximum value that passes through the PA bus, only one complete fetching and one complete writing of the whole image would be needed. This turns in a reduction in the number of memory accesses from 2 257 Q^2 to 2 256 Q^2, incrementing the computational speed by 0.004%. The speed increment is thus negligible with respect to the additional hardware complexity that this solution would require.

Appendice C

Improving the Data-Bus Efficiency

When the number of PEs is smaller than the number of elements in the data set, as generally happens in Image Processing applications, a mechanism must be provided to virtualize the PA. The *External Virtualization* mechanism [14] is a widely used [87, 74, 81, 37, 42] technique for PE virtualization, generally implemented on low-cost SIMD architectures whose simple PEs only own a small amount of memory. The computation is serialized in windows: the PA is loaded from the external memory with a sub-window of the data set; then the computation is performed until a special instruction (hereinafter referred to as `UPDATE` to reflect the one used in the PAPRICA system) is reached; and finally the results are stored back again into the external memory. These steps are iterated until all the sub-windows have been processed. Then the first sub-window is reloaded again into the PA and the computation is resumed until the next `UPDATE` operation is found.

The main problem of this kind of virtualization mechanism is due to the fact that when the PA is loaded with a sub-window of the data set, the PA border processors cannot access their complete neighborhood. In fact, after the execution of a single instruction involving a 3×3 neighborhood, the values stored in the PA border processors are meaningless. If more 3×3 neighborhood-based operations are executed, then also the PEs whose neighbors contain meaningless data cannot produce valid results. Thus only a portion (hereinafter referred to as the *Validity Area*, VA) of the data processed by the PA contains significant data. The next windows that will be loaded into the PA will be partially overlapped with the previous ones, in order to correctly evaluate the results which in the previous computation were invalid.

C.1 Performance Analysis

This section presents the performance analysis of a generic SIMD system, following the scheme shown in figure C.1.

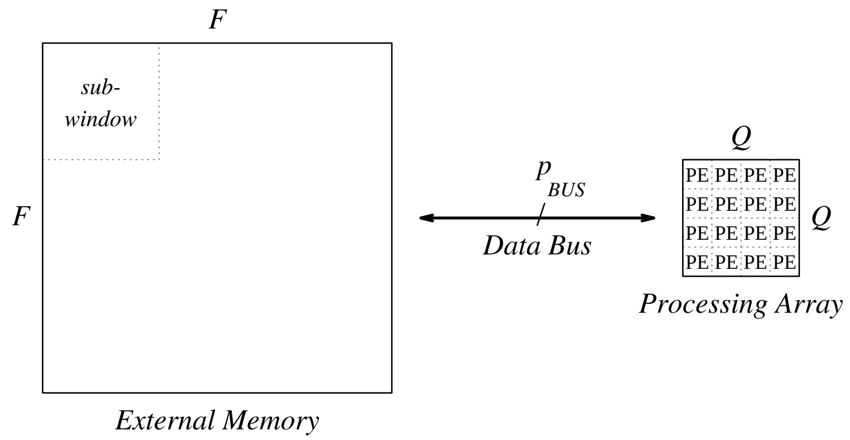


Figura C.1. A system implementing the External Virtualization Mechanism.

Let's assume a system composed of a $Q \times Q$ PA connected to the external memory (logically organized as in figure 3.4.a) through a bus with parallelism p_{BUS}; the basic transfer operation duration is T_M, while the PE instruction duration is T_C.

Let's assume that the data set to be processed by the system is a 2-dimensional array ($F \times F$) of elements with word parallelism[1] w. Parameters p_L and p_S represent the number of times the bus is used to transfer a single element of the data set, and are defined as the smallest integers greater than the ratio between the data set word parallelism and p_{BUS} during loading and storing operations respectively:

$$p_L = \left\lceil \frac{w_L}{p_{\mathrm{BUS}}} \right., \quad \text{and} \quad p_S = \left\lceil \frac{w_S}{p_{\mathrm{BUS}}} \right., \tag{C.1}$$

where subscripts L and S stand for *loading* and *storing*.

The generic application that is executed by the system is composed of L instructions, G of them being *graphic operations*, namely involving the access to the PE's 3×3 neighborhood. The remaining $L - G$ operations are *logical operations* performed using as input only the data stored in each PE internal memory.

As a consequence of the application of graphic operations, as already shown in section 3.4, the VA is reduced. In order to avoid that the VA is completely shrunk, the sequence of program instructions is interleaved with n_{upd} `UPDATE` operations. Their presence splits the program into segments (here defined `UPDATE` *blocks*) which are executed in sequence on the whole data set, as they were independent programs. The determination of the number and position of `UPDATE` operations may influence the performance (in terms of processing speed) in a deeper way than the choice of the processing algorithm itself. The determina-

[1] In Image Processing applications the data set word parallelism represents the number of bits per pixel.

tion of an automated procedure for the optimal position of `UPDATE` operations have been analyzed in [14], where the determination of the processing speed (as a function of the above introduced parameters) is also presented.

According to [14], the processing speed S_{pr} is given by:

$$S_{pr} = \frac{\left(Q - 2\,\frac{G}{n_{upd}}\right)^2}{(\overline{p_L} + \overline{p_S})\; Q^2\; T_M\; n_{upd} + L\; T_C}\,. \tag{C.2}$$

Note that p_L and p_S depend on the specific elaboration performed by each `UPDATE` block. In equation (C.2) p_L and p_S have been substituted by $\overline{p_L}$ and $\overline{p_S}$ to indicate their average values within a given application.

The processing speed is in fact a function of parameters that depend on the system technology and architecture (T_M, T_C, Q), on the specific computational task (L, G), on the specific coding[2] of the data set $(\overline{p_L}, \overline{p_S})$, and on the number of `UPDATE` instructions (n_{upd}).

According to the theoretical analysis developed in [14] for the determination of the optimal value for n_{upd} (as a function of the architectural and application-dependent parameters), it follows that

$$\left[n_{upd}\right]_{opt} = 6\,\frac{G}{Q}\,. \tag{C.3}$$

Combining equations (C.2) and (C.3), the processing speed in the case of optimal positioning of `UPDATE` operations can be expressed by:

$$S_{pr} = \frac{4}{9}\;\frac{Q^2}{6\,(\overline{p_L} + \overline{p_S})\; Q\; G\; T_M + L\; T_C}\,. \tag{C.4}$$

Assuming $T_M \simeq T_C$, as far as L is negligible with respect to $6\,(\overline{p_L} + \overline{p_S})\; Q\; G$ (for example when Q is sufficiently large), the dependence of the processing speed on the length of the program is negligible.

[2]For example a *compressed* format vs a *plain* format of the input and temporary data sets.

Since the system parameters (Q, T_M, T_C) are fixed by the physical architecture, and factors G and L are determined by the specific application, the processing speed can be increased by determining an efficient data coding: the tuning of parameters $\overline{p_L}$ and $\overline{p_S}$ for the optimization of the processing speed is the topic of the next section.

C.2 Incrementing the Hardware Efficiency

This section provides a way to improve the system performance (the processing speed) based on the maximization of the hardware efficiency. With the maximization of the PA efficiency it has been possible [14] to obtain equation (C.3) for the optimization of the number and position of `UPDATE` operations. Let's now focus on the maximization of the data bus efficiency.

The efficiency of the data bus η during transfer operations is defined as the ratio between the word parallelism of the data to be transferred and the number of bits actually transferred (which is given by the number of times the bus is used to transfer a single element of the data set times the data bus parallelism):

$$\eta \triangleq \frac{\text{data\_set\_parallelism}}{\text{number\_of\_transfers} \times \text{bus\_parallelism}} \leq 1 \ . \qquad \text{(C.5)}$$

As in the case above, let's distinguish between data bus efficiency during loading and storing operations:

$$\eta_L \triangleq \frac{w_L}{p_L \; p_{\text{BUS}}} \quad \text{and} \quad \eta_S \triangleq \frac{w_S}{p_S \; p_{\text{BUS}}} \ . \qquad \text{(C.6)}$$

It is easy to deduce that, according to equations (C.1) and (C.6),

$$\eta_L \begin{cases} = 1 & \text{if } \frac{w_L}{p_{\text{BUS}}} \text{ is an integer} \\ \\ < 1 & \text{if } \frac{w_L}{p_{\text{BUS}}} \text{ is not an integer} \end{cases} , \qquad \text{(C.7)}$$

and

$$\eta_S \begin{cases} = 1 & \text{if } \frac{w_S}{p_{\mathrm{BUS}}} \text{ is an integer} \\ \\ < 1 & \text{if } \frac{w_S}{p_{\mathrm{BUS}}} \text{ is not an integer} \end{cases} \quad . \tag{C.8}$$

The maximization of the data bus efficiency is then reduced to the use of a data set word parallelism in transfer operations that is a multiple of the data bus parallelism p_{BUS}. For sake of simplicity, from now on η will be used instead of η_L and η_S, and w instead of w_L and w_S to parametrize a generic data transfer between the PA and the external memory.

Considering as an example the PAPRICA system, where $p_{\mathrm{BUS}} = 16$, when an `UPDATE` block operates on a binary image, the efficiency of the data bus η is considerably low. In fact η becomes:

$$\left[\eta\right]_{\mathrm{binary\_images}} = \frac{1}{16} \simeq 6\% \ . \tag{C.9}$$

As shown in equations (C.8) the data bus efficiency η can be incremented if all the data flowing through the bus (p_{BUS} bits each memory cycle) would held significant data. This can be achieved if the data transfer process is preceded by a data packing procedure, which stacks more elements of the original data set onto a single element of a new packed data set. The new ideal data set (still a 2D array) will contain a different number of elements with respect to the original one, and each element will have a word parallelism which is a multiple of p_{BUS}. Note that the new data set is obtained through a packing procedure, and not through a data compression procedure: data are only grouped and the data coding is not modified.

Data Reorganization and Packing

In order to simplify the comprehension of the following analysis, the processing of images is considered. Images are 2D pixel sets, where w indicates the image depth in bit/pixel.

Let us consider two constants $q_x,\ q_y \in N^+$ such that $q_x \times q_y \times w$ is as close as possible (but smaller or equal) to a multiple of p_{BUS}. The portion of the image memory which contains the $F_x \times F_y \times w$ processing frame can be rearranged and remapped onto another image memory area with dimensions $\left(\frac{F_x}{q_x}\right) \times \left(\frac{F_y}{q_y}\right) \times \left(q_x\ q_y\ w\right)$. This means that the number of sub-windows in which the external virtualization mechanism splits the image is sensibly reduced, but the major benefit lays in the reduction of factor G. In fact, as shown in figure C.2, through a single 3×3 neighborhood-based operation each PE has access to the information stored in a wider *logical neighborhood*[3], whose dimensions depend on the pixel position within the $q_x \times q_y$ sub-window. The *maximum logical neighborhood* accessible by every pixel with a single 3×3 neighborhood-based operation corresponds to a square of $(2\ q_x + 1,\ 2\ q_y + 1)$. The execution of G operations based on a 3×3 neighborhood on a packed image means that a $(2\ G\ q_x + 1,\ 2\ G\ q_y + 1)$ *logical neighborhood* is accessed.

Thus, assuming $q_x = q_y = q$, the relation between factor G when processing a plain or a packed image is:

$$\left[G\right]_{packed} = \frac{\left[G\right]_{plain}}{q}\ . \tag{C.10}$$

The same considerations apply to the number of UPDATE instructions $\left[n_{upd}\right]_{packed} =$

[3]Note that the *logical neighborhood* is different from the PE physical neighborhood which is fixed by the hardware architecture: the *logical neighborhood* refers to the data set neighborhood that can be accessed within a single operation.

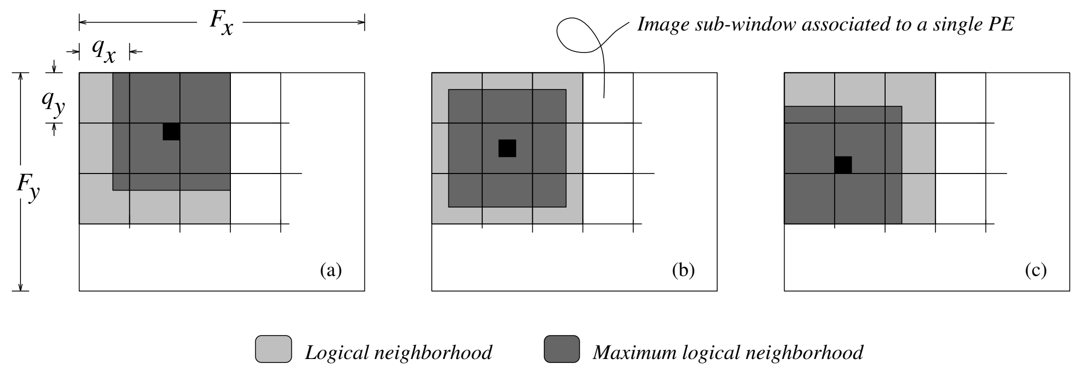


Figura C.2. A pixel in a packed image: the dimensions of its *logical neighborhood*, reachable through a single 3×3 neighborhood-based operation, depend on the pixel position within the $q_x \times q_y$ sub-window. Conversely, the *maximum logical neighborhood* is independent of the pixel position.

$\dfrac{\left[n_{upd}\right]_{plain}}{q}$, and to the linear frame dimension $\left[F\right]_{packed} = \dfrac{\left[F\right]_{plain}}{q}$. Moreover, since a q^2 image sub-window is associated to a single PE, each PE must repeat q^2 times the same elaboration (a few considerations and some optimizations that could shorten the final program length are presented in the following paragraphs), leading to the following relation:

$$\left[L\right]_{packed} = q^2 \, \left[L\right]_{plain} \, . \tag{C.11}$$

Assuming $p_L = p_S = 1$ for sake of simplicity, the processing speed is then expressed by

$$\left[S_{pr}\right]_{packed} = \frac{4}{9} \; \frac{Q^2}{\dfrac{12 \; Q \; G \; T_M}{q} + q^2 \; L \; T_C} \, , \tag{C.12}$$

instead of

$$\left[S_{pr}\right]_{plain} = \frac{4}{9} \; \frac{Q^2}{12 \; Q \; G \; T_M + L \; T_C} \, , \tag{C.13}$$

where n_{upd}, G, and L are referred to the application working on a plain image. Now, for sufficiently large values of q, $q^2 \; L \; T_C$ is no more negligible with respect to $\dfrac{12 \; Q \; G \; T_M}{q}$ in equation (C.12). The most important consideration is anyway due to the fact that when processing a packed image, the frame dimensions are reduced. The processing time, defined as

$$T_{pr} \triangleq \frac{(F)^2}{S_{pr}} \;, \tag{C.14}$$

is in fact given by:

$$\Big[T_{pr}\Big]_{packed} = \frac{\left(\frac{F}{q}\right)^2}{\Big[S_{pr}\Big]_{packed}} = \frac{9}{4}\;\frac{F^2}{Q^2}\;\left(\frac{12\;Q\;G\;T_M}{q^3} + L\;T_C\right) \tag{C.15}$$

instead of

$$\Big[T_{pr}\Big]_{plain} = \frac{F^2}{\Big[S_{pr}\Big]_{plain}} = \frac{9}{4}\;\frac{F^2}{Q^2}\;\Big(12\;Q\;G\;T_M + L\;T_C\Big)\;. \tag{C.16}$$

Equations (C.15) and (C.16) show that the higher q, the higher the benefit of processing a packed data set instead of a plain one. Anyway, it must be highlighted that the total time $T_{\rm tot}$ required for the processing

- matches $\Big[T_{pr}\Big]_{plain}$ in the case of a plain image, but
- is larger than $\Big[T_{pr}\Big]_{packed}$ in the case of a packed image, since two data reorganization procedures must be performed:

$$\Big[T_{\rm tot}\Big]_{plain} = \Big[T_{pr}\Big]_{plain} \;, \tag{C.17}$$

while

$$\Big[T_{\rm tot}\Big]_{packed} = T_{packing} + \Big[T_{pr}\Big]_{packed} + T_{unpacking}\;. \tag{C.18}$$

C.3 Data Reorganization on PAPRICA System

In the case of PAPRICA architecture (where $p_{\rm BUS} = 16$), when processing binary images ($F \times F \times 1$) the optimal choice is $q_x = q_y = 4$. The *maximum* *logical neighborhood* accessed when $G = 1$ is therefore 9×9. Table C.1 shows the dimensions of the *maximum logical neighborhood* accessed in the case of a plain image or a packed one.

Reduction Factor	Logical neighborhood (plain image)	Logical neighborhood (packed image)
$G = 0$	1×1	1×1
$G = 1$	3×3	9×9
$G = 2$	5×5	17×17
$G = 3$	7×7	25×25
$G = 4$	9×9	33×33

Tabella C.1. The *maximum logical neighborhood* in the case of a plain and packed image for $q = 4$.

Figure C.3 shows how the reorganization of a binary image takes place on PAPRICA system: a $Q \times Q \times p_{\mathrm{BUS}}$ sub-window is loaded from the external memory (area A) to the PA with a data bus efficiency $\eta_L = \frac{1}{p_{\mathrm{BUS}}}$; then data are reorganized and moved within the PE internal memories; finally a $\frac{Q}{q} \times \frac{Q}{q} \times p_{\mathrm{BUS}}$ subset of the PEs data is stored into the image memory (area B) with a higher data bus efficiency $\eta_S = \frac{q^2}{p_{\mathrm{BUS}}}$.

This behavior is possible because PAPRICA system allows different mappings between the PEs internal memories and the external memory during load and store procedures. Moreover, it is possible to set a `MARGIN` parameter to $\frac{q-1}{2\,q}\,Q$, so that only the central square array $\left(\frac{Q}{q} \times \frac{Q}{q}\right)$ of the PA values is saved back into the image memory.

The time needed to reorganize a $F \times F \times 1$ image, with $q = 4$, is given by the sum of:

(a) the loading time $F^2\, T_M$;

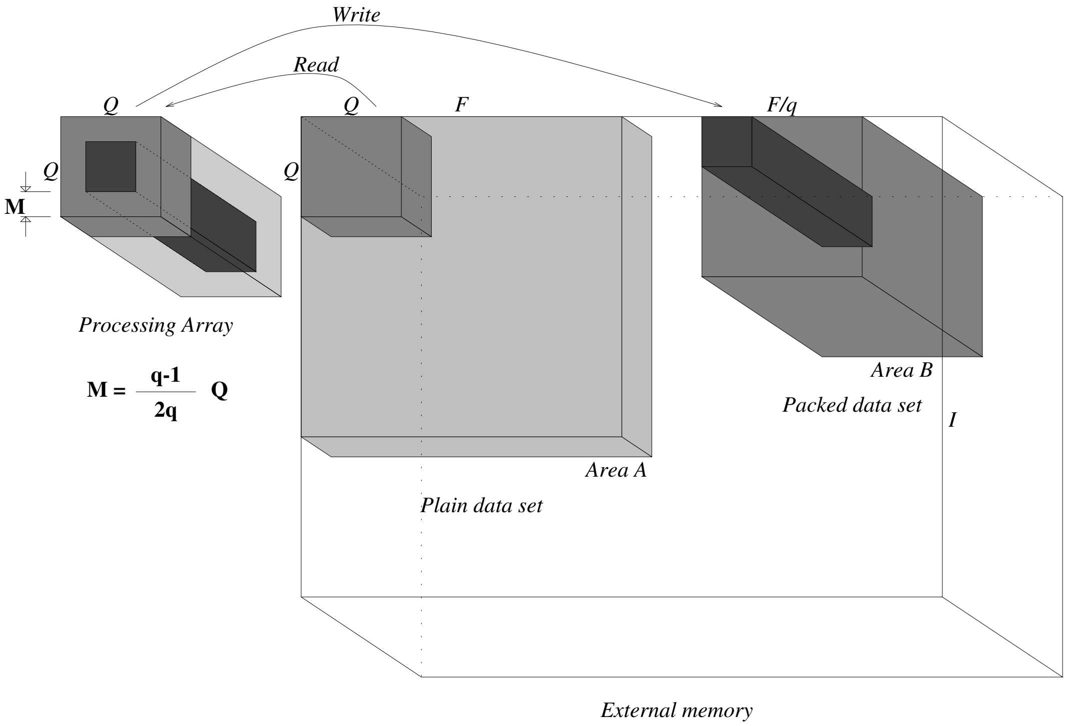


Figura C.3. The data reorganization procedure on PAPRICA system.

(b) the reorganization time $\left(\frac{F}{Q}\right)^2 L_r\ T_C$; and

(c) the saving time $\left(\frac{F}{4}\right)^2 T_M$:

$$T_{packing} = F^2\ T_M + \left(\frac{F}{Q}\right)^2\ L_r\ T_C + \left(\frac{F}{4}\right)^2\ T_M \simeq F^2\ \left(T_M + \frac{L_r\ T_C}{Q}\right)\ , \quad \text{(C.19)}$$

where L_r represents the number of Assembly instructions needed by the data reorganization routine.

Similar considerations apply to the data unpacking procedure, which requires the same amount of time to restore the resulting packed data set into a plain one:

$$T_{unpacking} = T_{packing}\ . \quad \text{(C.20)}$$

Performance Analysis on PAPRICA System

As an example, let us consider a single thinning [84] iteration, which in [7] is reduced to a sequence of 8 matchings with 3×3 patterns $\left(\left[G\right]_{plain} = 8\right)$, corresponding to a 17×17 *image neighborhood.* Assuming $Q = 16$, it corresponds to a program formed by 3 `UPDATE` blocks. After the data packing procedure (with $q = 4$), the VA reduction factor becomes $\left[G\right]_{packed} = 2$, and thus $\left[n_{upd}\right]_{packed} = 1$. Considering that $\left[L_r\right]_{plain} \simeq 50$ and $\left[G\right]_{plain} = 8$, the time required to filter a 256×256 image with a single thinning iteration on PAPRICA system ($Q^2 = 256$, $T_M = 350$ ns, $T_C = 350$ ns) is $\left[T_{1\_\mathrm{iter}}\right]_{plain} = 237$ ms, while $\left[T_{1\_\mathrm{iter}}\right]_{packed} = 13$ ms. The sum of the time $T_{packing}$ and $T_{unpacking}$ required for the two $1 \rightarrow q^2$ and $q^2 \rightarrow 1$ data reorganization procedures is about 2×36 ms $= 72$ ms, thus even the execution of a single iteration is sufficient to justify the data reorganization[4].

Table C.2 shows the execution timings of a different number of iterations of a thinning filter on a plain image $\left(\left[T_{\mathrm{tot}}\right]_{plain}\right)$, on a packed image $\left(\left[T_{\mathrm{tot}}\right]_{packed}\right)$, and the relative speed-up percentage, defined as:

$$\text{Speed-up} \triangleq 100 \left[\frac{\left[T_{\mathrm{tot}}\right]_{plain}}{\left[T_{\mathrm{tot}}\right]_{packed}} - 1 \right] . \tag{C.21}$$

In order to exploit this optimization criterion, a program has been developed for the automatic translation of a generic PAPRICA Assembly code working on binary data sets to its equivalent version working on packed data.

[4]A new data packing technique is currently under study and evaluation, based on a more efficient exploitation of PAPRICA hardware features: it should allow to decrease the data reorganization time to about 18 ms.

Iterations	$\left[T_{\mathrm{tot}}\right]_{plain}$	$\left[T_{\mathrm{tot}}\right]_{packed}$	Speedup %
# 1	237 ms	85 ms	180 %
# 2	475 ms	98 ms	380 %
# 4	712 ms	111 ms	540 %
# 8	950 ms	124 ms	660 %

Tabella C.2. Performance of a thinning filter working on a 256×256 plain and a 64×64 packed version of the same image.

The Automatic Code Conversion

This section presents an automatic procedure used to convert a PAPRICA Assembly program working on binary images into its equivalent version working on 16 bit packed images ($q = 4$).

According to equation (C.11), each single-bit instruction will be converted into a sequence of q^2 instructions. The following example shows how a single instruction is converted. Assuming that layer 0 in the plain image maps onto

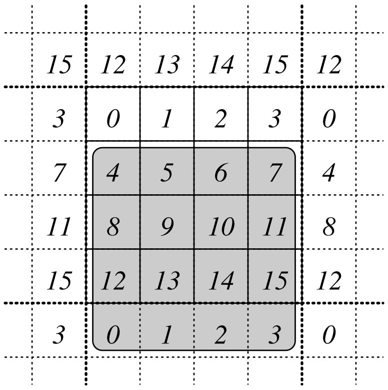


Figura C.4. A north translation in a packed image

layers 0÷15 in the packed image, layer 1 onto layers 16÷31 and so on, instruction `L1 = NMOV(L0)` (binary layer 1 is loaded with the north translation of layer 0) will be converted, according to figure C.4, into the following sequence:

```
L16 = (L4)          L24 = (L12)
L17 = (L5)          L25 = (L13)
L18 = (L6)          L26 = (L14)
L19 = (L7)          L27 = (L15)
L20 = (L8)          L28 = NMOV(L0)
L21 = (L9)          L29 = NMOV(L1)
L22 = (L10)         L30 = NMOV(L2)
L23 = (L11)         L31 = NMOV(L3)
```

Note that, since each PE memory contains a 4×4 logical neighborhood, the previous sequence is formed by 12 mere data transfers within the PE memory and 4 read operations from the PE neighborhood.

Without entering too much into the detail of the implementation, only the two following optimization criteria will be highlighted.

(a) Due to the specific implementation of PAPRICA instruction set, some complex instructions (such as the 8-direction dilation `EXP`) must be considered as a combination of 8 translations, and thus their equivalent version working on packed sets will be formed by 8×16 instructions, instead of only 16.

(b) On the other hand, instead of converting one instruction at a time, in some cases it is possible to convert a set of instructions, generating shorter programs working on packed sets.

As shown in equation (C.12), for sufficiently large values of q, the length of the program is no more negligible when processing packed sets, and thus the role of the code optimizer is of basic importance. Case (a) highlights that it is not worth to optimize the original code working on plain sets, since the optimizations cannot be transferred to the converted code. On the other hand, case (b) shows that sometimes it is possible to convert some specific code configurations into programs which are shorter than what would be obtained with an instruction-wise translation. The advantages highlighted by case (b) depend on the specific code segment to be converted, but it has been shown that it is possible to achieve a 60 % reduction in the number of instructions of the output code.

Appendice D

The Abingdon Cross Benchmark on PAPRICA System

D.1 Algorithm Definition

Since PAPRICA system is composed of single bit processing elements, the processing of binary images is preferred with respect to the handling of data structures with higher word parallelism. Moreover, due to the specific instruction set implemented on PAPRICA, high performances are obtained when the algorithm is based on morphological operations. Thus

(a) the first step of the filtering is a thresholding operation, which produces a binary image; Then

(b) a number of morphological operations are performed; and finally

(c) a thinning filter [84] is applied on the resulting image.

In the implementation of points *(b)* and *(c)* the optimization technique presented in Appendix C has been used: it is based on the consideration that the data bus, which links the program memory and the processor array, has a 16 bit parallelism, while the data flowing through it are binary, producing a data bus efficiency of about 6% only. The improvement of the data bus efficiency is the goal of such a technique, and it is achieved by a data packing procedure. The data packing, performed during the thresholding step (point *(a)*), reads from the image memory a $N \times N \times 1$ image and writes a $\frac{N}{4} \times \frac{N}{4} \times 16$ image. Besides incrementing the bus efficiency to 100%, it decreases the image linear dimensions from N to $\frac{N}{4}$, and it increases the pixels' neighborhood accessed by an individual read instruction [17].

D.2 Performance Analysis

The total computational time T is given by the sum of the contributions of the three steps described above: T_a, T_b, and T_c. Being T_M and T_C the memory access time and the processor cycle time respectively, each contribution is given by the sum of three components: loading time, processing time, and storing time.

$$T_a = N^2\ T_M + \frac{N^2}{Q^2}\ L_s\ T_C + \frac{N^2}{4^2}\ T_M\ , \tag{D.1}$$

where L_s assembly instructions are executed by the thresholding operation on each one of the $\frac{N^2}{Q^2}$ subwindows in which the image has been partitioned.

Steps *(b)* and *(c)* are then performed on the packed version of the image. Step *(b)* has been implemented as the sequence of one morphological opening, three closings, and four erosions, and consists of L_m assembly instructions. An individual thinning iteration on a non packed image (in the morphological implementation presented in [7]) has a margin value of 8, and consists of L_t assembly instructions. The optimal margin value (for $Q = 16$) when the discussed data packing procedure is used is 12 [17]. Thus the complete morphological filtering is done within a single Update Block, while a total of 2 Update Blocks can contain 3 iterations of the morphological thinning. Defining

$$L_x \triangleq \frac{L_m + 3\ L_t}{3} , \tag{D.2}$$

the total time required to perform steps *(b)* and *(c)* is given by

$$T_b + T_c = 3\ n_{\text{win}} \left[Q^2\ T_M + 4^2\ L_x\ T_C + \left(Q - 2\ G \right)^2\ T_M \right] , \tag{D.3}$$

where n_{win} represents the number of overlapping subwindows in which the image is partitioned. n_{win} is given by:

$$n_{\text{win}} = \left[\frac{N - 2G}{Q - 2G} \right]^2 . \tag{D.4}$$

The term 4^2 derive from the data packing procedure: for each memory reading or writing, 4^2 data are transferred in parallel on the data bus, and thus the elaboration must be repeated 4^2 times.

The total processing time for $N = 128$, $Q = 16$, $L_x \simeq 50$, and $L_s \simeq 30$ is:

$$T \simeq 177600\ T_M + 397600\ T_C . \tag{D.5}$$

Considering the current values for T_M and T_C, it becomes about 200 ms. The second prototype of the system runs the same task in about 110 ms.

The quality factor, as defined in equation (3.23), becomes $QF = 6.4\ 10^2$ for the first prototype, and $QF = 1.1\ 10^3$ for the second prototype. Table 3.5 reports the results shown in [80] for the non-commercial machines only, together with the results for the two versions of the PAPRICA system.

The complete output of the Abingdon Cross Benchmark is depicted in figure D.1.

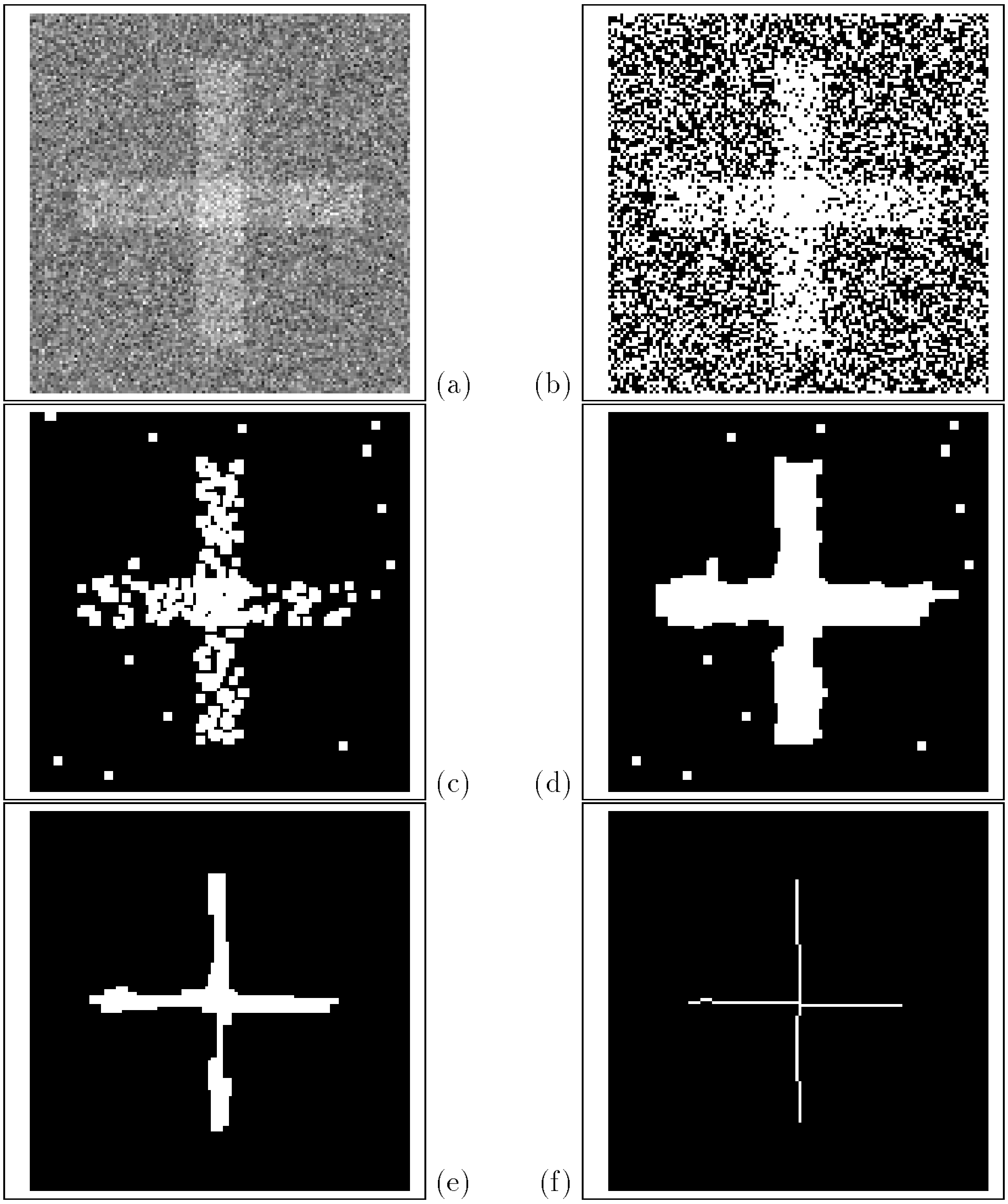


Figura D.1. The Abingdon Cross benchmark on a 128×128 image: *a*) input image; *b*) thresholded image; *c*) after one morphological opening with the elementary anisotropic structuring element (3×3); *d*) after one morphological closing with a 7×7 anisotropic structuring element; *e*) after four morphological erosions with the elementary anisotropic structuring element (3×3); *f*) after three thinning iterations.

Appendice E

The PAPRICA-3 System

E.1 The Linear Solution

As shown in figure E.1, the core of the PAPRICA-3 processor is a dedicated SIMD cellular architecture based on a linear array of Q identical 1-bit Processing Elements (PEs). The array as a whole is connected to an external image memory via a bidirectional Q-bit bus, therefore each memory read or write operation transfers a complete vector of Q pixels, 1 bit per pixel, at a time. The *Image Memory (IM)* contains one or more *Images*, each one being made of H lines of length Q pixels. The rationale behind this system is that the size of the PA matches exactly the width of the input image, i.e. the width of the imager device. This solution reduces the PE virtualization mechanism problem, which has been proven to be a critical design issue [57]. Therefore the value of Q will range from a minimum of 128 to 1024 if the implementation constraints, namely the cost, will allow. The PA processes one full image line per machine cycle, whose duration is T_C (about 10 ns). Since the PEs implement a *bit-serial approach*, at each cycle the PA executes morphological and boolean operations

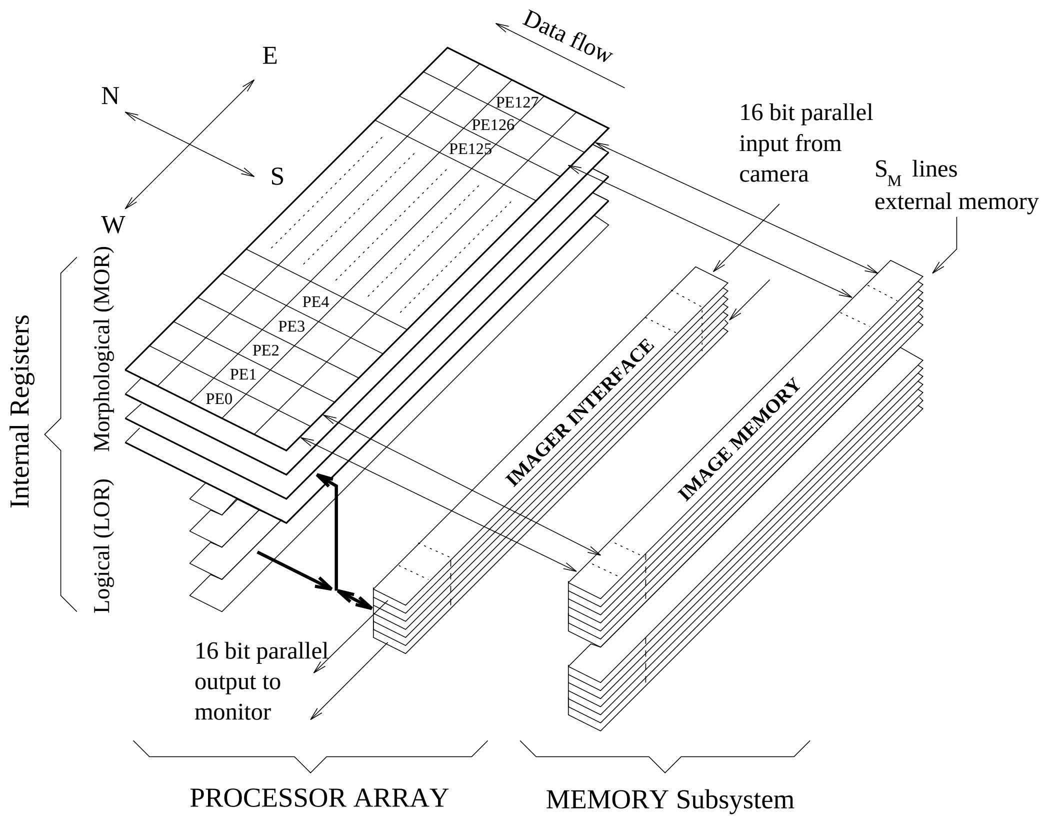


Figura E.1. Structure of the Processor Array

just on 2 or 3 layers at a time, while for more complex computations several machine cycles are needed both for operand loading and for processing (e.g. 8 bit add or subtract requires 8 cycles, while 8 bit multiplications need up to 64 cycles).

Data are transferred into the PEs internal registers, processed, and explicitly stored back into memory according to a RISC-oriented processing paradigm. Since the instruction set is based on morphological operators, the result of an operation depends, for each processor, on the values of pixels in a given neighborhood (5×5). Data from `EAST` and `WEST` directions may be obtained by direct connection with neighboring PEs while all other directions correspond to data of previous (`N`,`NE`,`NW`) or future (`S`,`SE`,`SW`) data lines. For this reason a number of processor registers (*Morphological Registers (MOR)*) have a structure which is more complex than that of a simple memory cell and are in reality composed of 5 1-bit cells with a `S`→`N` shift register connection. Besides the MOR, each PE owns also a number of *Logical Registers (LOR)* which may be used for intermediate storage and for all operations which do not require the use of neighboring values.

The system comprehends also a Serial-to-Parallel I/O Device, called *Imager Interface*, which can be connected to a conventional camera. While a line is processed, the Imager Interface automatically loads the following image line from the camera. At the end of the processing, the PA stores the results back into the Imager Interface (on different bit-planes) and loads in parallel the following image line. During the data acquisition process, the Imager Interface behaves like a shift-register, loading the data serially from the camera, and outputs the processed data serially to a monitor.

It is well known [14] that, due to the restricted neighborhood accessed by the PEs, a program needs to be split into different blocks, and each block must be applied to the whole image, one line at a time. An algorithm for the optimal partitioning of an Assembly program is presented in [14]. In order to speedup the program fetching from the Program Memory, the system includes also a *Writable Control Store (WCS)* (internal to the PA) that is used to store each block of instructions, for a faster access.

Two logical interprocessor communication mechanisms are available for exchanging information among PEs which are not directly connected. The first one (the Flag Evaluation Network, FEN), shown in figure E.2, is global the whole PA and provides two `SET/RESET` Flags, reflecting the status of the cen-

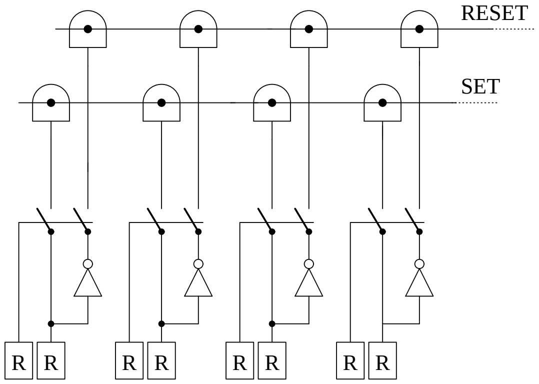


Figura E.2. Flag Evaluation Network (FEN)

tral pixel all PEs: all '1' or all '0', respectively. These two flags are used in control flow constructs handled by the PEs.

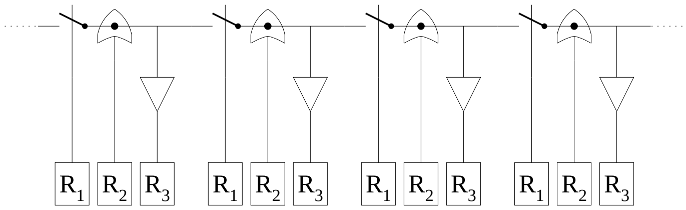


Figura E.3. Interprocessor Communication Network (ICN): for each PE, Register R_1 drives its ICN switch; Register R_2 sends its value over the network; and Register R_3 collects the resulting value from the network

The second one (the Interprocessor Communication Network, ICN), shown in figure E.3, allows global and multiple communications among components of different subsets of the PA (clusters of PEs) and its interconnection topology is programmable. In fact, each PE drives a switch that enables or disables the communication between them; the PEs can thus be dynamically grouped into clusters in which each PE can broadcast its values to the whole cluster within a single instruction. This feature can be extremely useful in algorithms involving *seed-propagation* techniques, and in the emulation of pyramidal (hierarchical) processings.

E.2 Image Memory Organization

As shown in figure E.5, the PA is tightly interfaced to the Image Memory via a bidirectional bus. Bus parallelism must match the number Q of PEs, in order to exploit the full bandwidth of the communication channel. For smaller (e.g. embedded) systems, a multiplexed bus can also be used but this impairs the overall performance. Due to the large parallelism (at present $Q = 128$ to 1024), a number of 32-bit memory modules is used. All modules share the same address lines and they therefore span the same address space (currently 64K to 1M words of Q bits each).

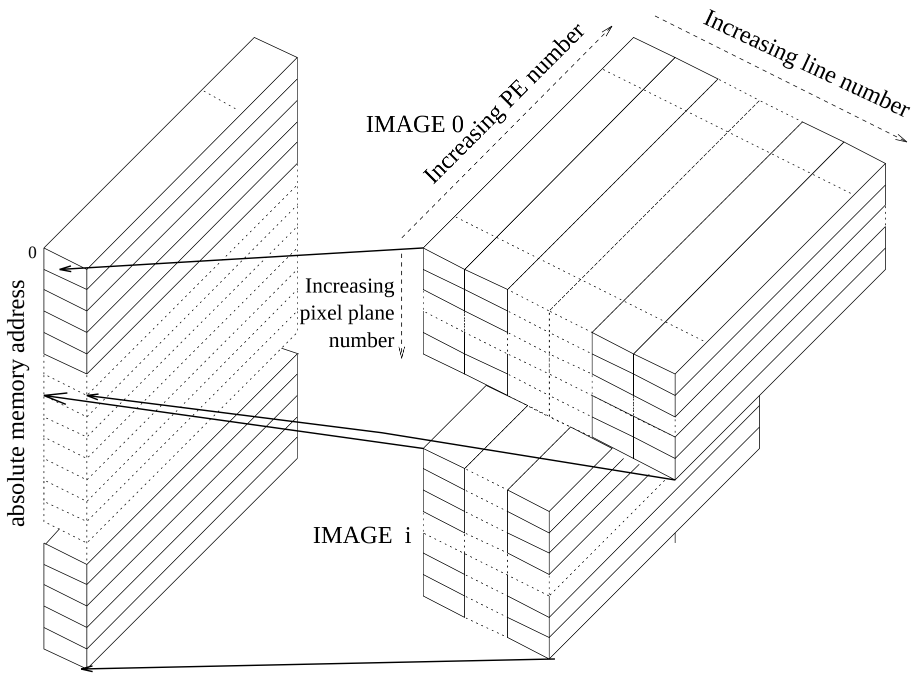


Figura E.4. Structure of the Memory Allocation

From the physical point of view, the IM can be seen as individually addressable lines of Q bits (*absolute line address*). However, this raw memory is organized into more complex logical structures, as shown in figure E.4, which may vary at run-time. As an example, lines can be arranged into *bit-planes* (i.e. a collection of adjacent lines) or into gray-scale or color images (i.e. a collection of bit-planes). Mapping between logical data structures and physical addresses is performed by a *Controller* (CT) external to the PA (see figure E.5).

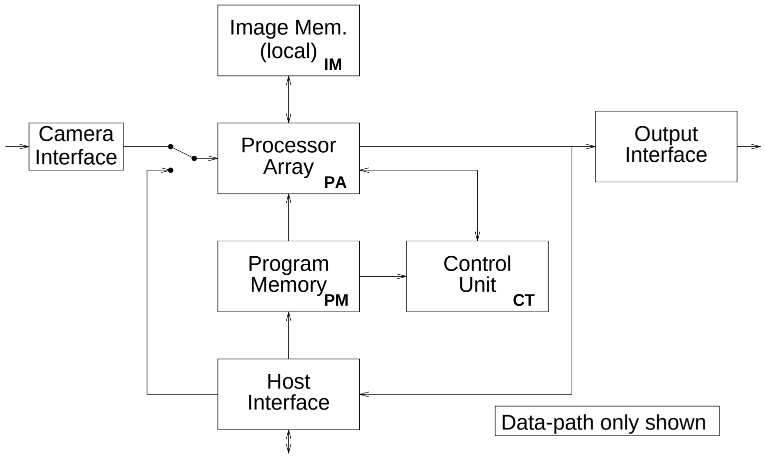


Figura E.5. Internal architecture of the PAPRICA-3 system

The memory speed is still a system bottleneck, since the internal clock period can be as small as 10 ns, while the cycle time of unexpensive memory modules is usually 3 to 5 times higher. This implies that memory transfer instructions shall insert a number of wait cycles.

According to the applications foreseen for the system, a different memory space can be envisaged. A small, low-cost embedded system can have as few as 32K$\times$64 bits, where a total of sixtyfour 8-bit images, 64×64 pixels of size can be stored. Larger systems can have as much as 8M$\times$256 bits, where up to forty 24-bit full color images, 2048×1024 pixels of size can be stored.

A large memory space is envisaged for high-definition images (e.g. for real-time HDTV applications). This provides much higher performance at the expense of a higher cost.

E.3 Instruction Set

Beside *Initialization*, *Memory Transfer*, and *Flow Control* Instructions, PAPRICA-3 features a *Morphological* Instruction Set based on a reduced 5×5 neighborhood [88]. Morphological Operators are parallel instructions which operate on a given "neighborhood" of each PE, and are based on *Matching Operators* derived from the *hit-or-miss* transform proposed by [88] and on its extension to pyramidal architectures.

A *Matching Template* is a 5×5 pixels ternary pattern ('1', '0', or 'don't care'), against which the neighborhood of each PE is compared for matching. The result of the operator is a boolean value which can either be true or false if the neighborhood matches or not the template, respectively.

An elementary PAPRICA instruction is the concatenation of a unary match operator `MOP` applied to a given bit-plane, with a binary *Logical Operator* `LOP`. The latter is applied to the `MOP`'s output and to another bit-plane of the central pixel.

The overall result may be either stored or accumulated (either OR-ed or AND-ed) into a destination register. The complete syntax of a generic morphological instruction is either of:

```
Rd   =  [MOP](Rs1) [LOP (Rs2)] [%EN]
Rd  |=  [MOP](Rs1) [LOP (Rs2)] [%EN]
Rd  &=  [MOP](Rs1) [LOP (Rs2)] [%EN]
```

where `Rd`, `Rs1`, and `Rs2` are three PE registers, namely the destination, the `MOP`'s and the `LOP`'s second input, respectively. `MOP` is a binary coding of the matching template, while `LOP` indicates one of eight boolean functions (e.g. NOT, AND, OR, etc.). The three versions of the syntax refer to direct storage, OR-accumulation and AND-accumulation, respectively.

The optional flag `%EN` acts as a Write Enable Mask, in the sense that, when present, the writing (or accumulating) process is conditioned to the corresponding bit of the mask.

The instruction set gives also provision for handling (i.e. computing and using) the three global flags and the interprocessor communication network. The flow control instructions can behave differently, according to the flag status.

E.4 System Implementation

A first prototype of the system is currently being designed, to be implemented on a set of two ASIC chips. The first one (a full-custom design) will incorporate an array of either 64 or 128 PEs. The second one (a standard-cell, semi-custom design) will incorporate all the circuits for the controller CT.

Two or more PA chips can be cascaded to build an array of as many PEs as required by the application. Currently a value of Q between 64 and 1024 is envisaged. The PA chips, together with a number of commercial SIMM memory modules (currently $\frac{Q}{32}$ chips with 32 bits parallelism), the controller, imager and video interfaces and the host interface (or the host itself in an embedded system) can be incorporated together on a single PCB board (size will depend on the desired system size).

The controller chip handles all transfers to/from the memory, executes program flow control instructions, handles the synchronization and data transfer to/from a host computer (DSP) and synchronizes the imager and the video.

The current target technology is a double metal 1 μm CMOS. With that technology an internal clock rate or 100 MHz (i.e. cycle time $T_C = 10$ ns) is expected, by properly designing the PA chip. Pipelining and systolic techniques will be extensively used. A memory transfer cycle time T_M of 50 ns (i.e. a transfer rate of up to 5 Gbit/s for $Q = 256$ can also be obtained).

A reduced performance prototype, with no internal pipelining and lower speed memories, is also envisaged. This system will have an internal clock rate of 20 MHz and a memory cycle time T_M of 150 ns.

In the future more performing technologies (e.g. either 0.7 μm or 0.5 μm, triple metal) will provide an internal clock rate of up to 200 MHz and a number of PEs per chip up to 1024.

A Multi Chip Module (MCM) implementation is foreseen for further implementations, when very high performance is required in a very small space.

ALGORITHM	Bit per pixel	Small system (64 PE, 50/150 ns)	Large system (256 PE, 10/50 ns)
Contouring of B/W image	1	≈ 0.8 ns/pel	≈ 0.04 ns/pel
Skeleton of B/W image	1	$\approx 200 - 400$ ns/pel	$\approx 10 - 20$ ns/pel
Pattern Matching (5×5)	1	$\approx 0.8 - 1.6$ ns/pel	≈ 0.04 ns/pel
Matrix Summation	8	≈ 7 ns/pel	≈ 0.4 ns/pel
3×3 pixel Average	8	≈ 30 ns/pel	≈ 1.6 ns/pel
5×5 pixel Average	8	≈ 60 ns/pel	≈ 3.2 ns/pel
Optical Flow (max. ± 8 pels per direction, full search)	8	≈ 5.0 μs/pel	≈ 80 ns/pel

Tabella E.1. Estimated performance figures of PAPRICA-3 system (values between brackets are Q, T_C/T_M)

E.5 Performances

Thanks to the proposed architecture, PAPRICA-3 fully exploits two main factors: the massive parallelism of the PA and the large width of the memory bus. Furthermore both factors are matched to the size of the imager, since from preliminary analyses this seems a good compromise among system complexity, speed, chip pinout limitations, and flexibility. The feasibility of the system and a forecast of its performances have been obtained through a software system simulator, currently running on Unix workstations and PCs.

The expected performance of the proposed architecture depend mainly on the specific application. The tasks exploiting the maximum performance are "cellular" or "morphological" algorithms, which require limited communications among PEs which are far apart from each other. A few performance figures for simple examples are presented in table E.1, assuming a worst case and a best case system composed respectively of 64 and 256 processing elements, with a processing cycle time of 50 ns (10 ns) and a memory cycle time of 150 ns (50 ns).

Appendice F

Removing the Perspective Effect

Considering the system configuration depicted in figure 5.16, the reorganization process replaces the one-to-one image transfer procedure needed to load the image from the frame grabber memory into the low-level processor memory. Since this transfer is always necessary, the discussed procedure introduces only a small computational overhead due to a more complex pixel mapping between the original image and the reorganized one. Moreover, it can be performed with a look-up-table (LUT), as in the current MOB-LAB implementation, allowing to reduce this computational overhead to a minimum.

The reorganization process is aimed to remove the perspective effect from the road image, producing a new image which represents the road as it were seen from above. This is accomplished by applying a function that associates to each pixel of the reorganized image its corresponding pixel in the original one. For this purpose, at first a function derived from a *ray-tracing* algorithm [76] has been studied to make a perspective projection of a 3D scene on a twodimensional screen. The removal of the perspective effect, then, is accomplished simply by applying its inverse function to the image acquired from the camera.

For the following discussion it is useful to define two Euclidean spaces:

- $\mathcal{W} \in E^3$ representing the $3D$ world space (*world-coordinate system*), where the real scene (or a model) is defined;
- $\mathcal{I} \in E^2$ representing the $2D$ image space (*screen-coordinate system*), where the 3D scene is projected.

The following two sections present the mapping between these two spaces, represented by the two function f and g:

$$\begin{cases} f : (u, v) \in \mathcal{I} & \rightarrow \quad (x, y, z) \in \mathcal{W} \\ g : (x, y, z) \in \mathcal{W} & \rightarrow \quad (u, v) \in \mathcal{I} \end{cases} . \tag{F.1}$$

F.1 The Relationship between $\mathcal{I}$ and $\mathcal{W}$

Mapping the $\mathcal{W}$ space to the $\mathcal{I}$ space

In order to generate a 2D view of a 3D scene, a *viewpoint*, *viewing direction*, *aperture*, and *resolution* must be specified [76]. Let us make the following assumptions:

- *viewpoint*: the camera is placed in $C = (l, d, h) \in \mathcal{W}$;
- *viewing direction*: the optical axis $\hat{o}$ is determined by the following angles:
  - $\overline{\gamma}$: the angle formed by the projection (defined by versor $\hat{\eta}$) of the optical axis $\hat{o}$ on the plane $z = 0$ and the x axis (as shown in figure F.1.a);

– $\overline{\theta}$: the angle formed by the optical axis $\hat{o}$ and versor $\hat{\eta}$ (as shown in figure F.1.b);

- *aperture*: the camera angolar aperture is 2α;
- *resolution*: the camera resolution is $n \times n$.

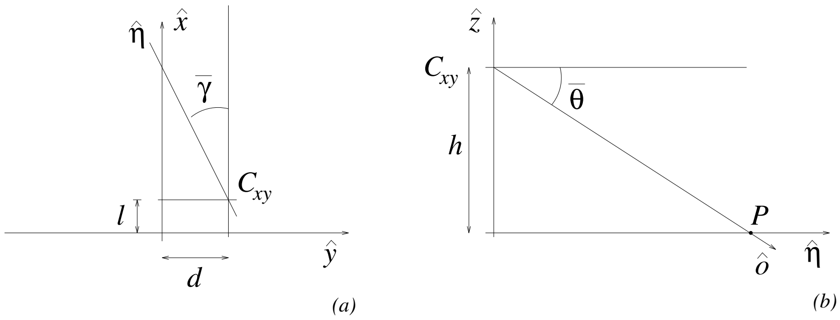


Figura F.1. *(a)* the xy plane in the $\mathcal{W}$ space and *(b)* the $z\eta$ plane, assuming the Origin translated onto the projection C_{xy} of C on $z = 0$

Now let us determine the intersection between each of the n^2 rays leaving the camera (within the viewing pyramid [76]) and the plane $z = 0$, assuming the region of interest belonging to this plane.

The n^2 rays are determined by $\theta(u)$ and $\gamma(v)$, obtained by the scanning of the camera angolar aperture. $\theta(u)$ and $\gamma(v)$ can vary as follows:

$$\begin{cases} \theta(u) = \left(\overline{\theta} - \alpha\right) + u \, \dfrac{2\alpha}{n-1} \\ \\ \gamma(v) = \left(\overline{\gamma} - \alpha\right) + v \, \dfrac{2\alpha}{n-1} \end{cases} , \tag{F.2}$$

where $u, v = 0, 1, 2, ..., n-1$ represent the coordinates in the $\mathcal{I}$ space.

The distance between the projection $C_{xy} \in \mathcal{W}$ of C on $z = 0$ and the intersection $P \in \mathcal{W}$ between the generic ray determined by the couple $\Big(\theta(u),$

$\gamma(v)\big)$ and the plane $z = 0$ is given by the following expression:

$$\text{distance}\,(C_{xy}, P) = \frac{h}{\text{tg}\,\Big[\theta(u)\Big]}\ . \qquad \text{(F.3)}$$

The projection of P on the x and y axes is given by:

$$\left\{\begin{array}{lcl} x(P) & = & \dfrac{h}{\text{tg}\,\Big[\theta(u)\Big]}\cos\gamma(v) + l \\ y(P) & = & \dfrac{h}{\text{tg}\,\Big[\theta(u)\Big]}\sin\gamma(v) + d \end{array}\right. . \qquad \text{(F.4)}$$

Using equations (F.2), the final mapping $f : \mathcal{I} \to \mathcal{W}$ as a function of u and v is then given by:

$$\left\{\begin{array}{lcl} x(u,v) & = & \dfrac{h}{\text{tg}\,\Big[(\overline{\theta}-\alpha) + u\,\dfrac{2\alpha}{n-1}\Big]}\cos\Big[(\overline{\gamma}-\alpha) + v\,\dfrac{2\alpha}{n-1}\Big] + l \\ y(u,v) & = & \dfrac{h}{\text{tg}\,\Big[(\overline{\theta}-\alpha) + u\,\dfrac{2\alpha}{n-1}\Big]}\sin\Big[(\overline{\gamma}-\alpha) + v\,\dfrac{2\alpha}{n-1}\Big] + d \\ z & = & 0 \end{array}\right. . \qquad \text{(F.5)}$$

Given the coordinates (u, v) of a generic point Q in the $\mathcal{I}$ space, equations (F.5) return the coordinates $(x, y, 0)$ of the corresponding point P in the $\mathcal{W}$ space (see figure 5.17). As an example, figure F.2.a shows a synthetic computer-generated texture that has been spread on the $z = 0$ plane of the $\mathcal{W}$ space. Figure F.2.b presents the result of the application of the *ray-tracing*-like algorithm defined by equation (F.5) on the previous image, using the set of real parameters of the camera installed on MOB-LAB and $n = 512$. This is the resolution that is generally available on any commercial camera.

It is easy to understand (even intuitively) that this transform causes an information loss, which is higher in the central region of figure F.2.b (corresponding to the top of figure F.2.a) and lower in the peripheral region of the same image (corresponding to the bottom of figure F.2.a).

Mapping the $\mathcal{I}$ space to the $\mathcal{W}$ space

Following similar algebraic and trigonometric manipulations, the inverse transform (the dual mapping) is given as follows. Given a generic point P with coordinates $(x, y, 0)$, let us now compute the angles formed by the optical axis $\hat{o}$ and segment $\overline{CP}$. Following the definitions of θ and γ given before, it is:

$$\left\{ \begin{array}{lcl} \gamma(x,y,0) & = & \operatorname{arctg}\left[\dfrac{y-d}{x-l}\right] \\ \theta(x,y,0) & = & \operatorname{arctg}\left[\dfrac{h\ \sin\gamma(x,y,0)}{y-d}\right] \end{array} \right. . \tag{F.6}$$

If $\gamma(x,y,0) \in \left[\overline{\gamma} - \alpha, \overline{\gamma} + \alpha\right]$ and $\theta(x,y,0) \in \left[\overline{\theta} - \alpha, \overline{\theta} + \alpha\right]$, then point P belongs to the viewing pyramid and its coordinates (u, v) in the $\mathcal{I}$ space are given by the following mapping $g : \mathcal{W} \to \mathcal{I}$:

$$\left\{ \begin{array}{lcl} u(x,y,0) & = & \dfrac{\gamma(x,y,0) - \left(\overline{\gamma} - \alpha\right)}{\dfrac{2\alpha}{n-1}} \\ v(x,y,0) & = & \dfrac{\theta(x,y,0) - \left(\overline{\theta} - \alpha\right)}{\dfrac{2\alpha}{n-1}} \end{array} \right. . \tag{F.7}$$

The reorganization process defined by equation (F.7) removes the perspective effect and recovers the texture of the $z = 0$ plane in the $\mathcal{W}$ space. It consists of scanning the array of pixels of coordinates $(x, y, 0) \in \mathcal{W}$ which form the reorganized image, in order to associate to each of them the corresponding value assumed by the point of coordinates $\big(u(x,y,0), v(x,y,0)\big) \in \mathcal{I}$.

Figure F.2.c shows the application of the reorganization process (defined by equation (F.7)) to the synthetic image of figure F.2.b. In this case the resolution chosen ($n = 128$) is determined as a good trade-off between information loss and processing time: the higher the resolution, the lower the information loss, but the higher the time required for its successive processing. Note that, as shown in figure F.2.c, the lower portion of the reorganized image is undefined.

Figure F.2.d shows that the undefined area can be divided into two regions: the height of the dark grey one is given by the specific camera position with respect to the x axis, while the left and right areas are due to the angolar aperture: the smaller the angolar aperture, the larger these areas.

As an example, figure F.2.f shows the effect of the reorganization procedure applied on the original frame shown in figure F.2.e: it is clearly visible that in this case the road markings width is invariant within the whole image. In this case the resolution chosen ($n = 128$) is determined as a good trade-off between information loss and processing time: the higher the resolution, the lower the information loss, but the higher the time required for its subsequent processing.

F.2 The Low-Level Processing

Identifying Road Markings

Thanks to the data reorganization, it is possible to determine the road markings position by means of an extremely simple morphological filter, which can be efficiently performed by massively parallel SIMD systems. The definition of a *"Road Marking"* used in this work is the following: a road marking in the $z = 0$ plane of the $\mathcal{W}$ space (i.e. in the reorganized image) is represented by a quasi-vertical bright line of constant width painted on a dark region (the road). Thus the pixels belonging to a road marking have a brightness value higher than their left and right neighbors. The simplest filter for road marking detection is thus based on the comparison among the brightness values of horizontally neighboring pixels.

For each image line $x = 0, 1, ..., n-1$, every pixel $P = (x, y, 0)$ compares its brightness value $b(x, y, 0)$ with its left and right ones: $b(x, y-m, 0)$ and

$b(x, y+m, 0)$, with $m \geq 1$. A new pixel value $r(x, y, 0)$ is then computed according to the following rule:

$$r(x,y,0) = \begin{cases} \Big(b(x,y,0) - b(x,y-m,0)\Big) + \Big(b(x,y,0) - b(x,y+m,0)\Big), \\ \qquad \text{if}\Big(b(x,y,0) \geq b(x,y-m,0)\Big) \& \Big(b(x,y,0) \geq b(x,y+m,0)\Big) \\ 0 \quad \text{otherwise} \end{cases} . \tag{F.8}$$

According to equation (F.8),

$$r(x,y,0) \neq 0 \implies \begin{cases} r(x,y-m,0) = 0 \\ r(x,y+m,0) = 0 \end{cases} . \tag{F.9}$$

The choice of m depends on the road markings width, which is assumed to be known. From now on, we will consider $m = 2$.

The resulting image[1] is shown in figure F.2.g.

Image Enhancement and Binarization

Due to different illumination conditions (e.g. in presence of shadows), the road markings may have different brightness, yet maintaining their superiority relationship with their horizontal neighbors. Thus a simple threshold seldom gives a satisfactory binarization and consequently an image enhancement filter is required, as well as an adaptive binarization. Exploiting the property expressed by equation (F.9), the left and right neighbors of a road marking line assume a zero value in the filtered image, as shown in figure F.2.h (obtained from fig-

[1]The rule described in equation (F.8) can be expressed in a more compact and simple way thanks to grey-tone mathematical morphology notations [61]: the new pixel value $r(x, y, 0)$ is given by:

$$r(x,y,0) = \begin{cases} d(x,y,0) - b(x,y,0) & , \quad \text{if } \; d(x,y,0) \geq b(x,y,0) \\ 0 & , \quad \text{otherwise} \end{cases} \tag{F.10}$$

where $d(x, y, 0)$ is defined as the grey-tone dilation [61] of the original brightness image by the following binary structuring element: $\boxed{\bullet}\boxed{}\boxed{\bullet}\boxed{}\boxed{\bullet}$.

ure F.2.g by giving a grey color to the zero values). Thus the execution of a few iterations (say h) of the following rule[2]

$$e^{(i+1)}(x,y,0) = \begin{cases} \max\Big(e^{(i)}(x+1,y,0), e^{(i)}(x-1,y,0), \\ \quad e^{(i)}(x,y+1,0), e^{(i)}(x,y-1,0)\Big), & \text{if } e^{(i)}(x,y,0) \neq 0 \\ 0 \quad \text{otherwise} \end{cases}, \tag{F.11}$$

starting from $e^{(0)}(x,y,0) = r(x,y,0)$, generates an enhanced image, as shown in figure F.2.i.

The binarization is performed by means of an adaptive threshold:

$$t(x,y,0) = \begin{cases} 1 & \text{if } e^{(h)}(x,y,0) \geq \dfrac{m(x,y,0)}{k} \\ 0 & \text{otherwise} \end{cases}, \tag{F.12}$$

where $m(x,y,0)$ is the maximum value computed within a given $c \times c$ neighborhood, and k is a constant. The result of the binarization of figure F.2.i considering $h = 10$, $k = 2$, and $c = 11$ is presented in figure F.2.j.

Figure F.2.k shows the representation in the $\mathcal{I}$ space of the binarized result of figure F.2.j, while figure F.2.l presents its final superimposition onto the original image. The high quantization visible in the lower region of these images is the effect of the choice of a medium resolution (128×128) for the reorganized image.

[2] Note that equation (F.11) can be efficiently expressed as a dilation with the following binary structuring element $\begin{array}{|c|c|c|}\hline & \bullet & \\ \hline \bullet & \bullet & \bullet \\ \hline & \bullet & \\ \hline \end{array}$, contextualized (see [62]) to the state of the central pixel.

F.3 Qualitative and Quantitative Performance Analysis

Beside being easily implementable on any SIMD massively parallel processor based on a morphological computational paradigm, this sequence of filters has the great advantage of resolving efficiently the main problem of vision-based road following systems: the presence of shadows on the road region. Thanks to the data reorganization, the road markings are represented by quasi-vertical lines; the discussed sequence of filters in fact enhances and extracts the vertical lines and discards the other brightness discontinuities whose gradient has a non-horizontal direction. Moreover this sequence of filters is extremely robust with respect to the noise caused by sunny blobs on a shaded road.

The discussed filter is currently being integrated and optimized on PAPRICA massively parallel architecture. Table F.1 presents a few preliminary timings of an optimized implementation on PAPRICA, obtained by a theoretical analysis of the algorithm complexity. The complete acquisition and low-level processing of a single frame takes less than 100 ms, thus allowing the processing of about 10 frames per second.

Operation	Time
Acquisition of a 512×512 image, 8 bit/pixel	in real time
Data Transfer and Reorganization with LUT (128×128)	40 ms
SIMD road marking detection (128×128)	55 ms

Tabella F.1. Preliminary timings on PAPRICA system

Figures F.3, F.4, F.5, F.6, and F.7 present a few results of the processing of images acquired under different conditions. Each figure presents:

(a) the original image (512×512 pixel, 8 bit/pixel),

(b) the reorganized one (128×128 pixel, 8 bit/pixel),

(c) the result of the road markings detection in the previous image (128×128 pixel, 1 bit/pixel), and

(d) the superimposition of the previous result onto the original image (512×512 pixel, 8 bit/pixel).

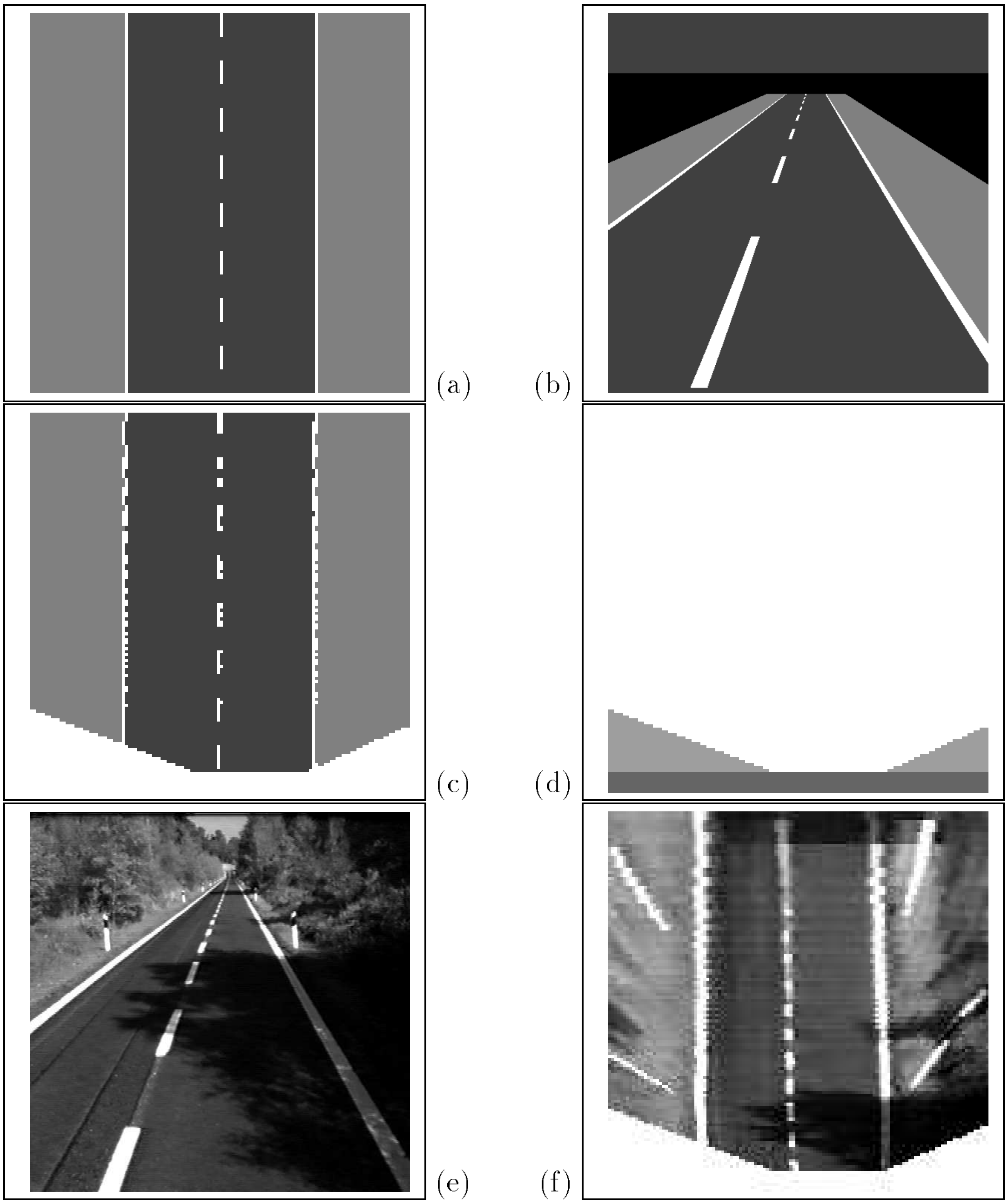


Figura F.2. *a*) The model image (this image has been rescaled to a square for displaying purposes); *b*) the projection of the previous model on the $\mathcal{I}$ space; *c*) the result of the reorganization procedure applied on the previous image; *d*) the two undefined regions in the reorganized image; *e*) the original image; *f*) the reorganized image.

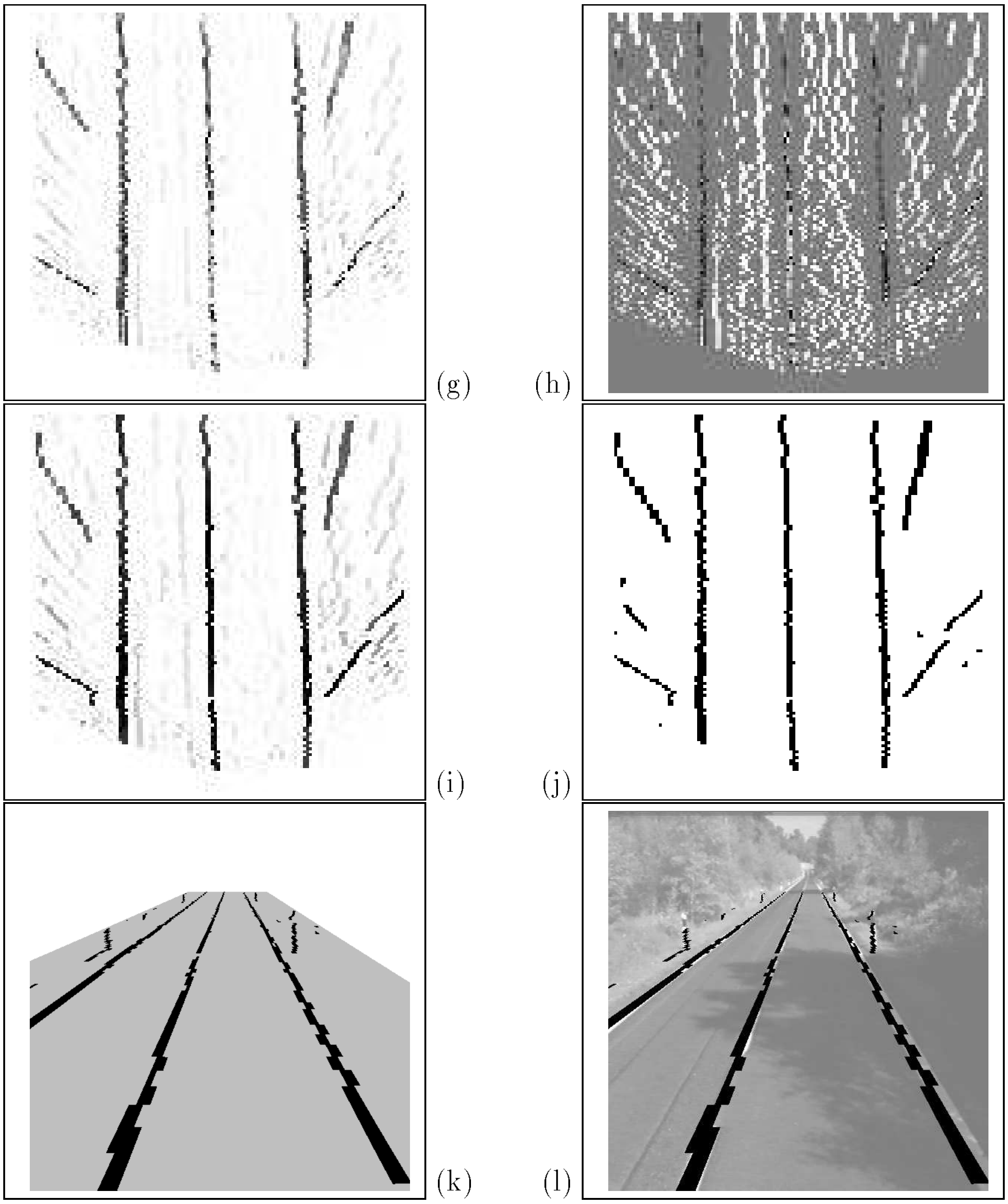


Figura F.2. (continued) g) The filtered image; h) the grey pixels represent zero values in the previous image; i) the enhanced image; j) the binarized image; k) the projection of the previous result on the $\mathcal{I}$ space; l) the superimposition of the previous image onto the original one.

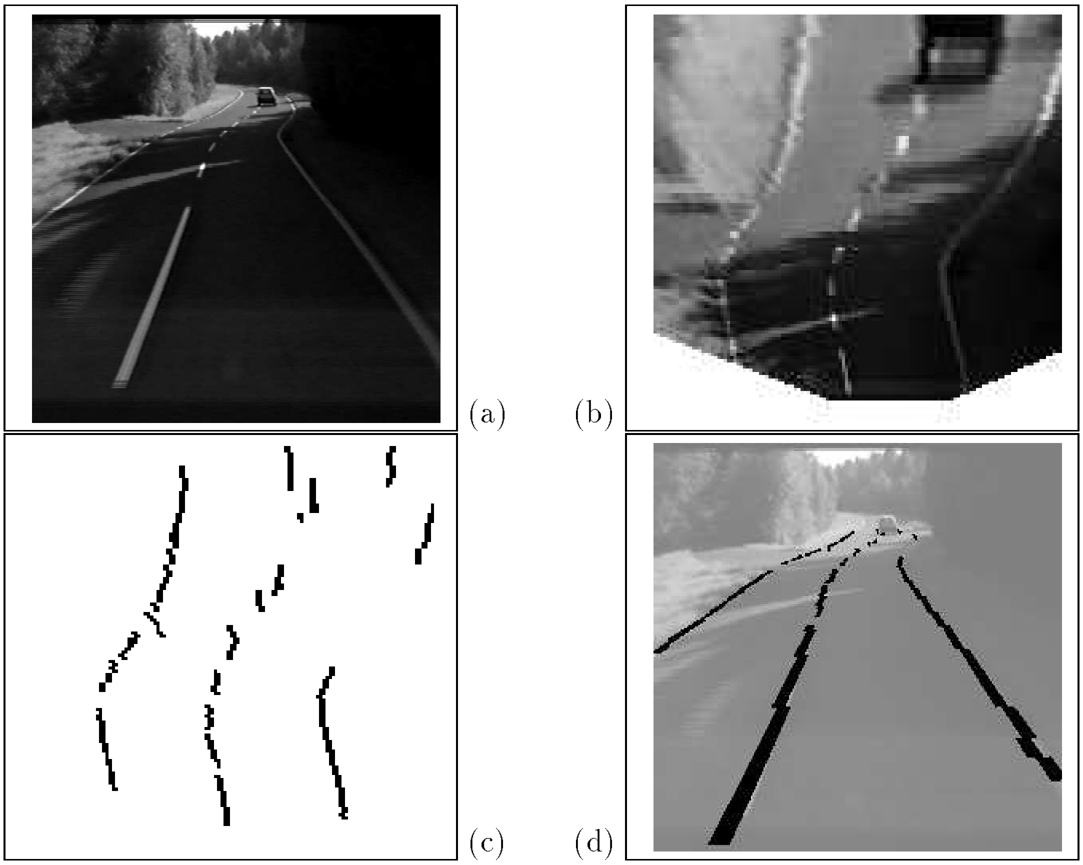


Figura F.3. A curved road with shadows: *a*) original image; *b*) reorganized image; *c*) road markings; *d*) superimposition of the previous result onto the original image.

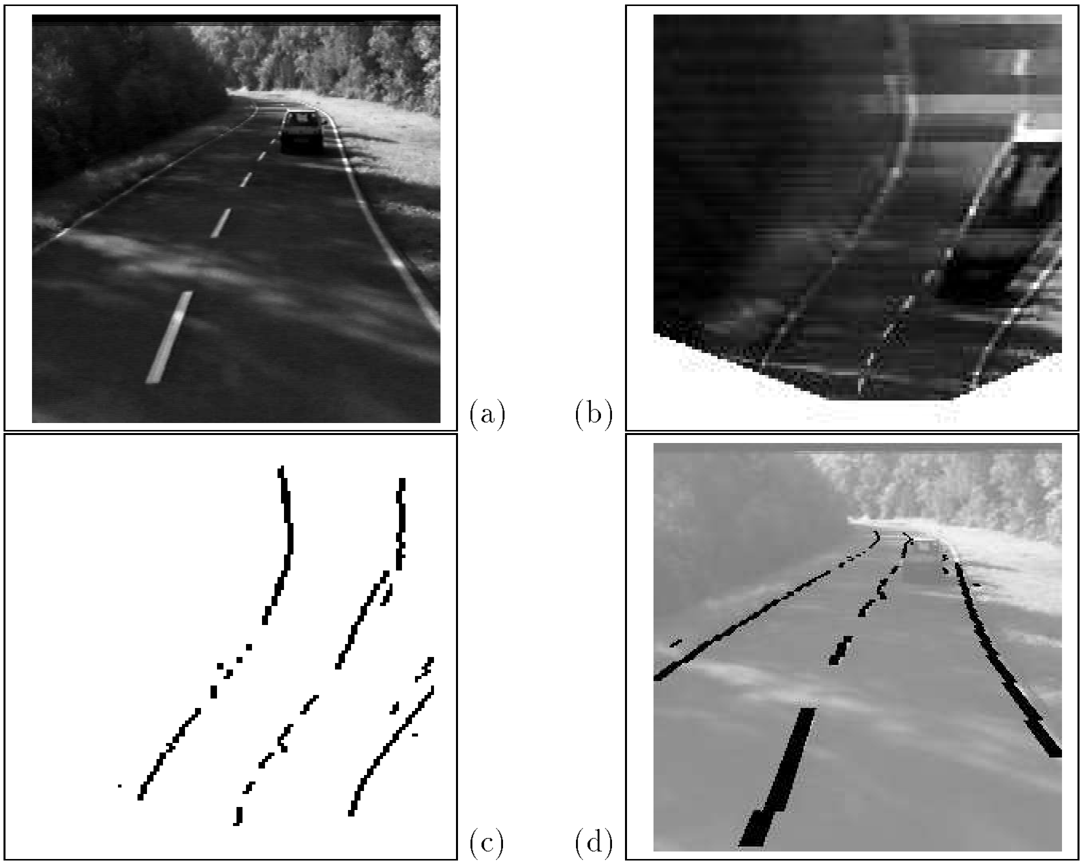


Figura F.4. A curved road with shadows and other vehicles: *a*) original image; *b*) reorganized image; *c*) road markings; *d*) superimposition of the previous result onto the original image.

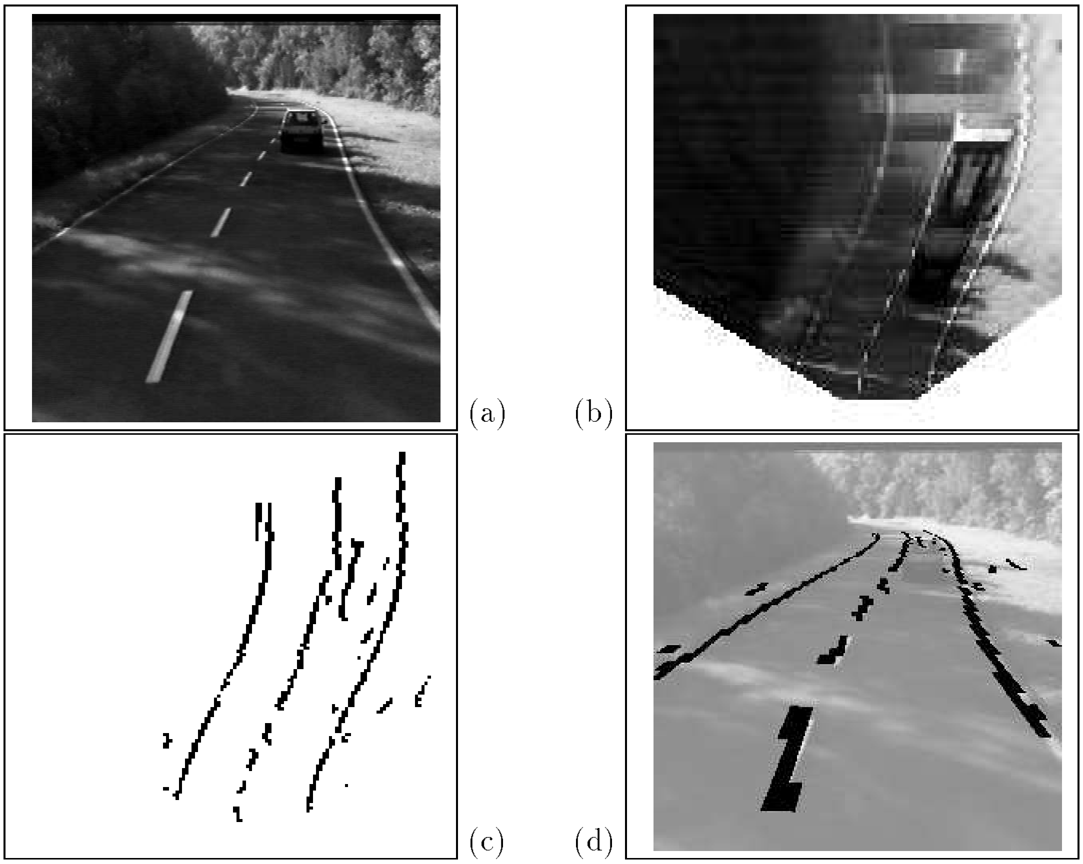


Figura F.5. The result of the processing of the previous image with different scale values: *a*) original image; *b*) reorganized image; *c*) road markings; *d*) superimposition of the previous result onto the original image.

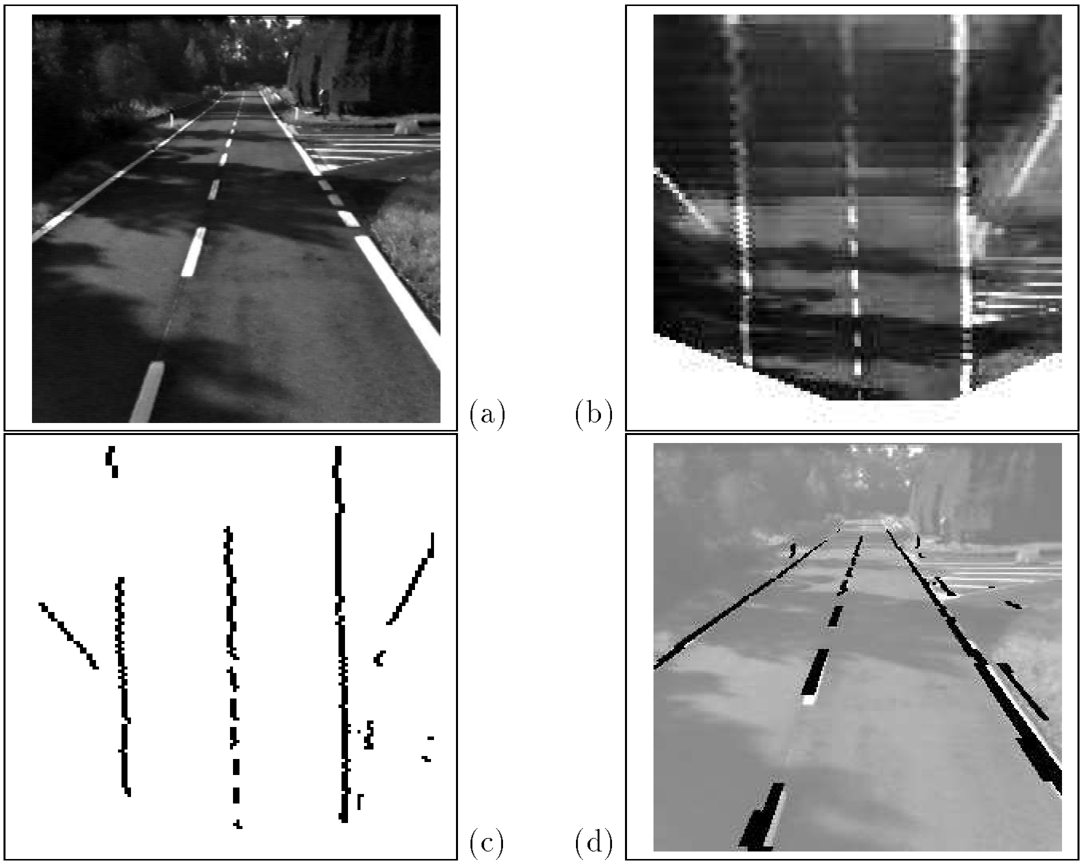


Figura F.6. A straight junction with shadows: *a*) original image; *b*) reorganized image; *c*) road markings; *d*) superimposition of the previous result onto the original image.

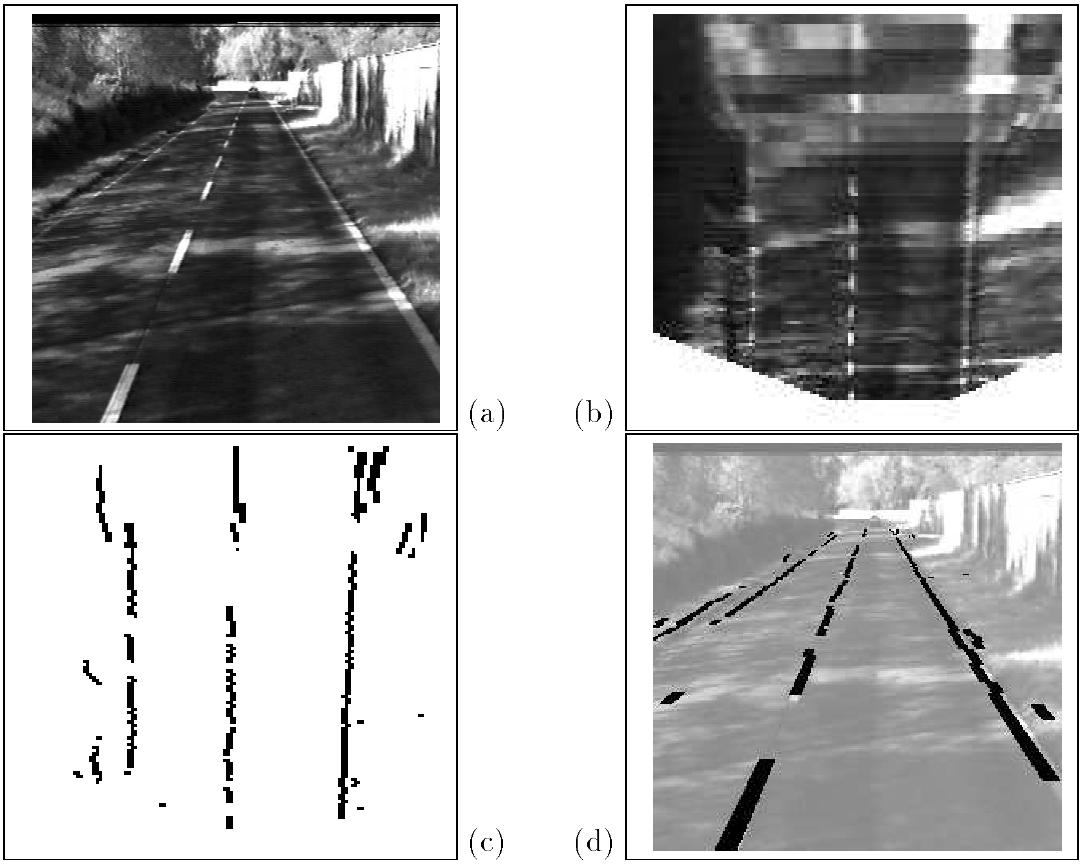


Figura F.7. A straight road with a lot of shadows and sunny blobs: *a*) original image; *b*) reorganized image; *c*) road markings; *d*) superimposition of the previous result onto the original image.

Appendice G

Lossy Image Compression on PAPRICA System

This Appendix investigates a novel approach to real-time image compression and recovery based on the use of massively parallel systems and data-parallel algorithms. In fact, the traditional sequential algorithms running on power-consuming systems based on expensive hardware platforms cannot meet the specific requirements of real-time processing on low-power and low-cost architectures. The algorithm discussed in this Appendix has the following advantages over the traditional transform-based compression algorithms (such as JPEG [106]):

- it can be efficiently expressed by a data-parallel computational model;
- it makes use of extremely simple operations only (additions, subtractions, and comparisons);
- the output quality can be traded for a higher processing speed and lower power consumption.

The algorithm discussed in this Appendix, in conjunction with conventional lossless serial compression algorithms [107], delivers quite promising performances in terms of both processing time and output quality.

G.1 System Description

The underlying basic structure of the discussed approach is common to the one utilized by the JPEG standard; it can be divided into 3 phases:

1. lossy filtering of the image data;
2. scanning of the 2D image array to produce a 1D data stream;
3. final compression of the data stream by means of traditional lossless techniques.

In this work only phases 1 and 2 have been redesigned according to the specific requirements presented in the previous section, while phase 3 is implemented utilizing the same techniques of the JPEG standard (serial lossless compression) such as Huffman coding, arithmetic coding, or LZW coding [107].

Phase 1, discussed in the following section, acts essentially as a low-level preprocessing filter, and has been tested on PAPRICA system, exploiting its specific features such as its massive parallelism, the possibility to handle hierarchical data structures efficiently, and its simple morphological instruction set. As shown in figure G.1, images are acquired from a camera, and preprocessed by PAPRICA system with simple low-level massively parallel computations (phase 1). The output data structure, still a 2D array of values, is scanned serially (phase 2) and then compressed by means of traditional sequential algorithms (phase 3) by PAPRICA front-end.

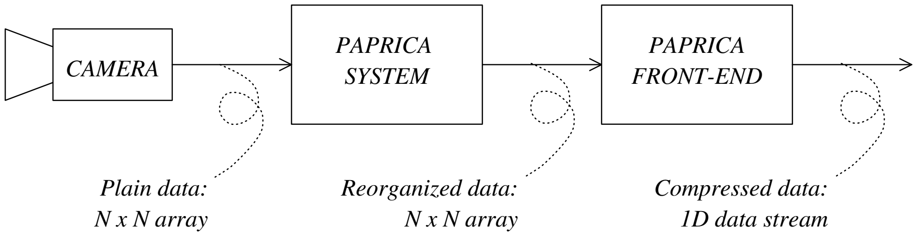


Figura G.1. Block diagram of data acquisition and processing with PAPRICA system

The Data Reorganization Procedure

This section describes the data reorganization process (phase 1), which is divided into two steps (lossless reorganization and non-linear filtering), and presents their hierarchical extension.

Let $\mathcal{F}^{N\times N} \subset \mathcal{N}^2$ be the set of all $N \times N$ frames, and consider the original image $P \in \mathcal{F}^{N\times N}$ formed by pixels $p_{i,j}$, with $0 \le i, j < N$. Image P is the input image to the lossy data reorganization procedure.

Assuming N even, the first transformation applied to image P, is a decomposition denoted by $\mathcal{D}(\cdot) : \mathcal{F}^{k\times k} \to \mathcal{F}^{k\times k}$:

$$P' = \mathcal{D}(P) \ , \quad \text{where } P, P' \in \mathcal{F}^{N\times N} \ , \tag{G.1}$$

and defined as follows:

$$p'_{i,j} = \begin{cases} p_{i,j} & \text{when both } i \text{ and } j \text{ are even} \\ \dfrac{p_{i-1,j} + p_{i+1,j}}{2} - p_{i,j} & \text{when } i \text{ is odd and } j \text{ is even} \\ \dfrac{p_{i,j-1} + p_{i,j+1}}{2} - p_{i,j} & \text{when } i \text{ is even and } j \text{ is odd} \\ \dfrac{p_{i-1,j-1} + p_{i+1,j+1}}{2} - p_{i,j} & \text{when both } i \text{ and } j \text{ are odd} \end{cases} \ , \tag{G.2}$$

for $0 \le i, j < N$, where $p_{i,j} = 0$, if either $i \ge N$ or $j \ge N$, thus assuming the image surrounded by zeros.

As shown in the following figure, image P' can be considered as composed by 4 interleaved images: S, X, Y, $Z \in \mathcal{F}^{\frac{N}{2}\times\frac{N}{2}}$, defined as:

$$
\begin{aligned}
S &= \left\{ s_{i,j} = p'_{2i,2j} \in P' \ , \ \text{ for } 0 \leq i,j < \frac{N}{2} \right\} \\
X &= \left\{ x_{i,j} = p'_{2i+1,2j} \in P' \ , \ \text{ for } 0 \leq i,j < \frac{N}{2} \right\} \\
Y &= \left\{ y_{i,j} = p'_{2i,2j+1} \in P' \ , \ \text{ for } 0 \leq i,j < \frac{N}{2} \right\} \\
Z &= \left\{ z_{i,j} = p'_{2i+1,2j+1} \in P' \ , \ \text{ for } 0 \leq i,j < \frac{N}{2} \right\}
\end{aligned}
\qquad \text{(G.3)}
$$

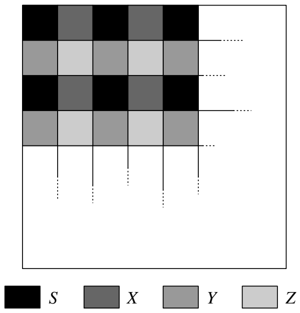


Image S represents the result of a mere 1:2 subsampling of image P, while each pixel of X, Y, and Z represents the local deviation from a linear signal in P (represented by the average between the pixel's neighbors in the x, y, and xy directions respectively). Obviously there exists an inverse transform that, starting from P', recovers P: $\mathcal{R}(\cdot) : \mathcal{F}^{k\times k} \to \mathcal{F}^{k\times k}$:

$$R = \mathcal{R}(P') \ , \quad \text{where } P', R \in \mathcal{F}^{N\times N} \ , \qquad \text{(G.4)}$$

with $R = P$. It is defined as:

$$
r_{i,j} = \begin{cases}
p'_{i,j} & \text{when both } i \text{ and } j \text{ are even} \\
\dfrac{p'_{i-1,j} + p'_{i+1,j}}{2} - p'_{i,j} & \text{when } i \text{ is odd and } j \text{ is even} \\
\dfrac{p'_{i,j-1} + p'_{i,j+1}}{2} - p'_{i,j} & \text{when } i \text{ is even and } j \text{ is odd} \\
\dfrac{p'_{i-1,j-1} + p'_{i+1,j+1}}{2} - p'_{i,j} & \text{when both } i \text{ and } j \text{ are odd}
\end{cases} \ , \qquad \text{(G.5)}
$$

for $0 \leq i, j < N$.

The advantage of image P' with respect to the original image P is that its histogram is much more polarized around low values, basically because $\frac{3}{4}$ of its pixels now represent only variations from a common (due to spatial correlation) local behavior (linear signal). Nevertheless, the effectiveness of the compression of image P' depends on the specific mapping of the 2D image array to a 1D data stream. In the next sections a possible mapping is presented and its effectiveness compared to the conventional left-to-right and top-to-bottom scanning.

Non-linear Filtering

The contributions of X, Y, and Z to the image histogram can be of two different kinds:

(a) low values, meaning small variations from the local linear behavior of the image brightness;

(b) high values, given by pixels which carry significant information such as edges (brightness discontinuities).

In order to improve furthermore the compression ratio, a non-linear filtering is performed on subimages X, Y, and Z. The reset to zero of small values in X, Y, or Z causes an obvious (but limited) information loss, but it allows a higher compression due to the considerable number of equal values (zeros) in the image histogram. Conversely, the preservation of high values tends to keep the information content unchanged.

Images X, Y, and Z are in fact filtered by a non-linear pointwise operator $\mathcal{T}_t(\cdot) : \mathcal{F}^{k\times k} \to \mathcal{F}^{k\times k}$ which is depicted in figure G.2:

$$\tilde{X} = \mathcal{T}_t(X) \quad , \quad \tilde{Y} = \mathcal{T}_t(Y) \quad , \quad \tilde{Z} = \mathcal{T}_t(Z) \quad , \tag{G.6}$$

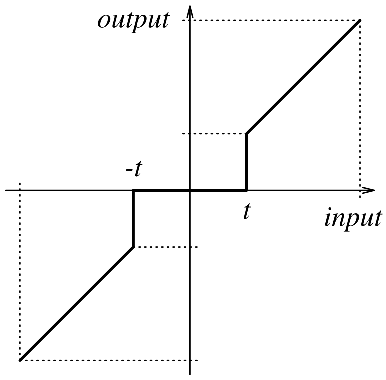


Figura G.2. Non-linear filter

defined as:

$$\tilde{x}_{i,j} = \begin{cases} x_{i,j} & \text{if } |x_{i,j}| \geq t \\ 0 & \text{if } |x_{i,j}| < t \end{cases} \quad \tilde{y}_{i,j} = \begin{cases} y_{i,j} & \text{if } |y_{i,j}| \geq t \\ 0 & \text{if } |y_{i,j}| < t \end{cases} \quad \tilde{z}_{i,j} = \begin{cases} z_{i,j} & \text{if } |z_{i,j}| \geq t \\ 0 & \text{if } |z_{i,j}| < t \end{cases} \tag{G.7}$$

for $0 \leq i, j < \frac{N}{2}$, which deletes the small (less than t) local deviations from a linear signal and preserves the information content carried by high brightness discontinuities (higher than t). This transformation is lossy, and the information content that is lost cannot be fully recovered by the application of the inverse transform $\mathcal{R}(\cdot)$. In fact $\mathcal{R}(\cdot)$, applied to image $\tilde{P}$ (composed of S, $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$), is able to reconstruct only an approximated version $\tilde{R}$ of the original image P.

Image Restoration

It is anyway important to note that a zero in a $\tilde{X}$, $\tilde{Y}$, or $\tilde{Z}$ subimage generally comes from the non-linear cut-off due to the application of $\mathcal{T}_t(\cdot)$. In this case, the reconstruction of the value of a generic pixel $p_{i,j}$ (with i or j odd) is affected by an approximation which may vary from $-t$ to $+t$. In order to keep the information on the error, during the recovering process two images are generated: the approximated version of the original image, $\tilde{R}$, and an error image, E, encoding the modulus of the maximal error which could affect the corresponding pixels in the recovered image $\tilde{R}$. The difference D between images P and $\tilde{R}$

$$D = P - \tilde{R} = \left\{ d_{i,j} = p_{i,j} - \tilde{r}_{i,j}, \text{for } 0 \leq i, j < N \right\} , \tag{G.8}$$

is such that $|d_{i,j}| \leq e_{i,j}$, for $0 \leq i, j < N$. Let us consider a generic image $K \in \mathcal{F}^{N \times N}$ such that $k_{i,j} = \tilde{r}_{i,j} + h_{i,j}$, for $0 \leq i, j < N$ and for any $|h_{i,j}| \leq e_{i,j}$. Due to the specific definition of K, the application of $\mathcal{D}(\cdot)$ and $\mathcal{T}_t(\cdot)$ to K produces image $\tilde{R}$:

$$\mathcal{T}_t \Big[\mathcal{D}(K) \Big] = \tilde{R} \; . \tag{G.9}$$

Thus, the couple of images $\tilde{R}$ and E represents in fact a set of images $K^{[i]}$ which may have generated set $\{S, \tilde{X}, \tilde{Y}, \tilde{Z}\}$. A further operation $\mathcal{C}(\cdot) : \mathcal{F}^{k \times k} \rightarrow \mathcal{F}^{k \times k}$ is thus needed to choose the one maximizing the *a-posteriori* probability, or, in other words, the one that best matches some specific criteria, determined by a-priori knowledge on the image. The assumptions used in this restoration process depend on the specific application: the restoration filters can range from image smoothing (continuity constraint), edge enhancement (weak continuity [11, 53] constraint), median filtering (to reduce noise spikes),...

Extension to a Hierarchical Process

Provided that $N = 2^n$, with $n \in \mathcal{N}$, and indicating with $S^{(0)}$ the original image P, decomposition $\mathcal{D}(\cdot)$ can be iterated n times: first on $S^{(0)}$ to produce $S^{(1)}$, $X^{(1)}$, $Y^{(1)}$, $Z^{(1)} \in \mathcal{F}^{\frac{N}{2} \times \frac{N}{2}}$, then on $S^{(1)}$ to produce $S^{(2)}$, $X^{(2)}$, $Y^{(2)}$, $Z^{(2)} \in \mathcal{F}^{\frac{N}{4} \times \frac{N}{4}}$, until $S^{(n)}$, $X^{(n)}$, $Y^{(n)}$, $Z^{(n)} \in \mathcal{F}^{1 \times 1}$. After the hierarchical decomposition, the non-linear filter $\mathcal{T}_t(\cdot)$ is applied n times with a set of different thresholds $t^{(1)}$,

$t^{(2)}$,... $t^{(n)}$. As an example, figure G.3 shows a block diagram of 2 iterations of the hierarchical decomposition and lossy filtering, while figure G.4 shows the corresponding hierarchical image recovery procedure. Like in the previous mono-resolution case, at each iteration of the recovery process two images are generated (the reconstructed image $\tilde{R}$ and the error image E):

$$\tilde{r}_{i,j}^{(k)} = \begin{cases} \tilde{p}_{i,j}^{(k+1)} & \text{when both } i \text{ and } j \text{ are even} \\ \dfrac{\tilde{p}_{i-1,j}^{(k+1)} + \tilde{p}_{i+1,j}^{(k+1)}}{2} - \tilde{p}_{i,j}^{(k+1)} & \text{when } i \text{ is odd and } j \text{ is even} \\ \dfrac{\tilde{p}_{i,j-1}^{(k+1)} + \tilde{p}_{i,j+1}^{(k+1)}}{2} - \tilde{p}_{i,j}^{(k+1)} & \text{when } i \text{ is even and } j \text{ is odd} \\ \dfrac{\tilde{p}_{i-1,j-1}^{(k+1)} + \tilde{p}_{i+1,j+1}^{(k+1)}}{2} - \tilde{p}_{i,j}^{(k+1)} & \text{when both } i \text{ and } j \text{ are odd} \end{cases} \tag{G.10}$$

for $0 \leq i,j < \dfrac{N}{2^k}$ and

$$e_{i,j}^{(k)} = \begin{cases} e_{\frac{i}{2},\frac{j}{2}}^{(k+1)} & \text{when both } i \text{ and } j \text{ are even} \\ \dfrac{e_{\frac{i-1}{2},\frac{j}{2}}^{(k+1)} + e_{\frac{i+1}{2},\frac{j}{2}}^{(k+1)}}{2} + t^{k+1}\delta\left[\tilde{p}_{i,j}^{(k+1)}\right] & \text{when } i \text{ is odd and } j \text{ is even} \\ \dfrac{e_{\frac{i}{2},\frac{j-1}{2}}^{(k+1)} + e_{\frac{i}{2},\frac{j+1}{2}}^{(k+1)}}{2} + t^{k+1}\delta\left[\tilde{p}_{i,j}^{(k+1)}\right] & \text{when } i \text{ is even and } j \text{ is odd} \\ \dfrac{e_{\frac{i-1}{2},\frac{j-1}{2}}^{(k+1)} + e_{\frac{i+1}{2},\frac{j+1}{2}}^{(k+1)}}{2} + t^{k+1}\delta\left[\tilde{p}_{i,j}^{(k+1)}\right] & \text{when both } i \text{ and } j \text{ are odd} \end{cases} \tag{G.11}$$

for $0 \leq i,j < \dfrac{N}{2^k}$, where the Dirac function $\delta(x)$ is defined as follows:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} . \tag{G.12}$$

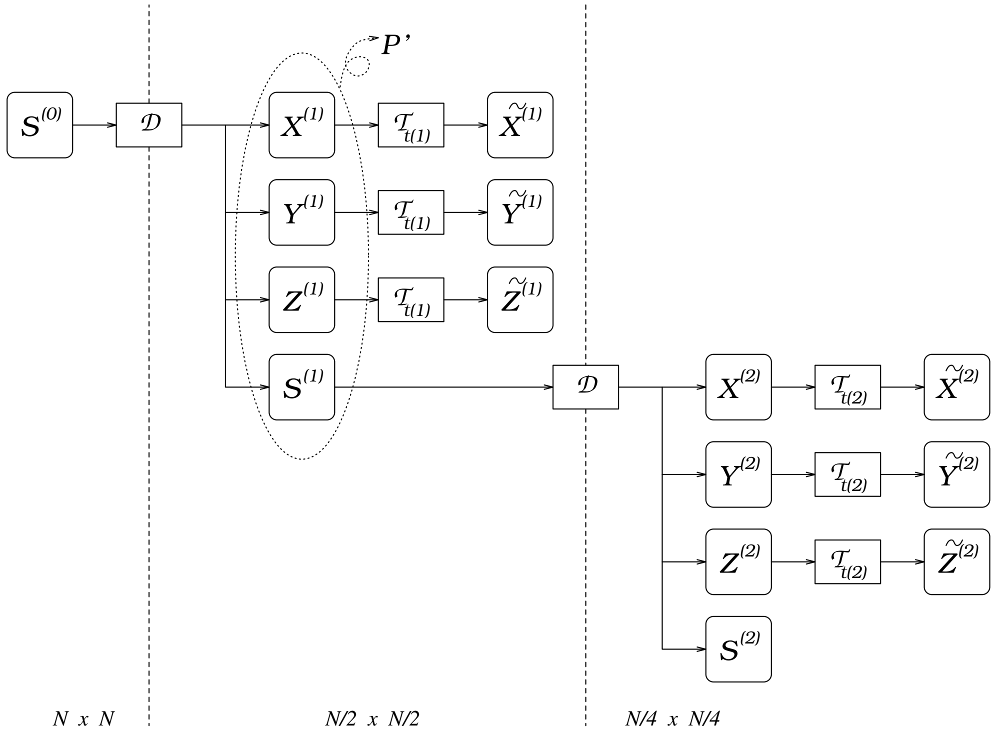


Figura G.3. Two iterations of the hierarchical process

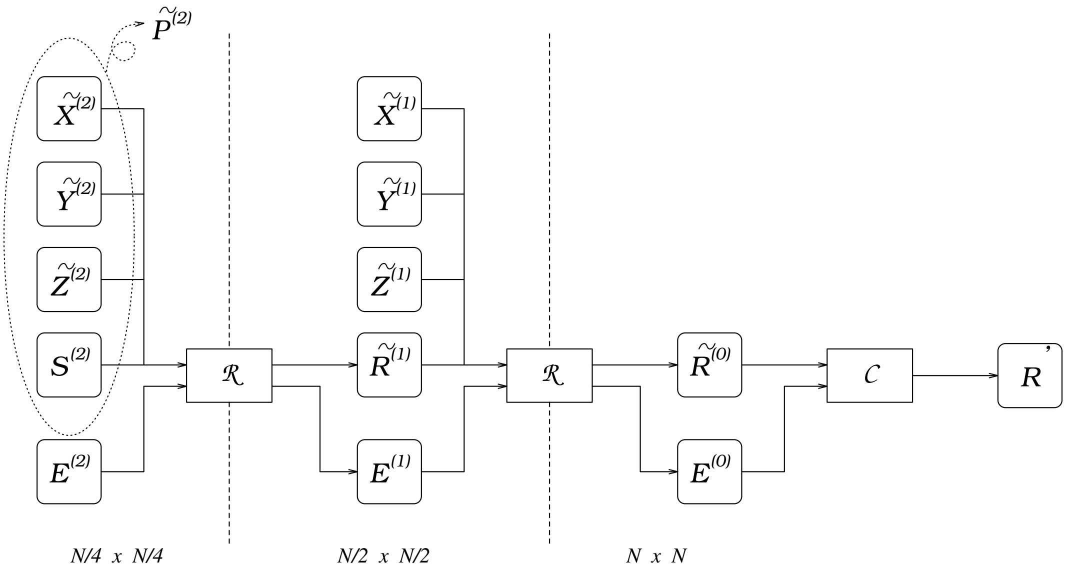


Figura G.4. Two iterations of the recovering procedure, followed by the restoration process

Expression (G.10) is the hierarchical extension of equation (G.5), while expression (G.11) has been obtained considering that the approximation in the determination of a pixel value is composed of two contributes:

- a function (average) of the interpolating values, and
- in case the exact information has been removed through the application of the threshold, the threshold value itself must be added.

Figure G.5 shows two iterations of the discussed procedure on a linear profile of an image.

As an example, figure G.8 shows the effect of 2 iterations of the lossless decomposition on the original image of figure G.6. Figure G.9 presents the set of 10 images generated by 3 iterations of the lossless decomposition followed by the non-linear filtering with thresholds $t^{(1)} = 60$, $t^{(2)} = 40$, and $t^{(3)} = 10$. The compression ratio achieved by the LZW algorithm on each image is also presented.

The choice of the set of thresholds, as well as the specific non-linear filter, are key points in the lossy process. In [73] the subjective quality of the recovered images has been correlated to the number of hierarchical iterations and to the set of thresholds in order to determine an automatic algorithm for their determination as a function of both the desired output quality and the required compression. Different non-linear filters can be used, depending on the specific characteristics of the images to be compressed. For example a filter which introduces a high quantization (small number of bits/pixel) to represent images $X^{(1)}$, $Y^{(1)}$, and $Z^{(1)}$ and reduces the quantization (increases the number of bits/pixel) for the successive hierarchical iterations is currently under evaluation and testing for videoconference applications (for facial images).

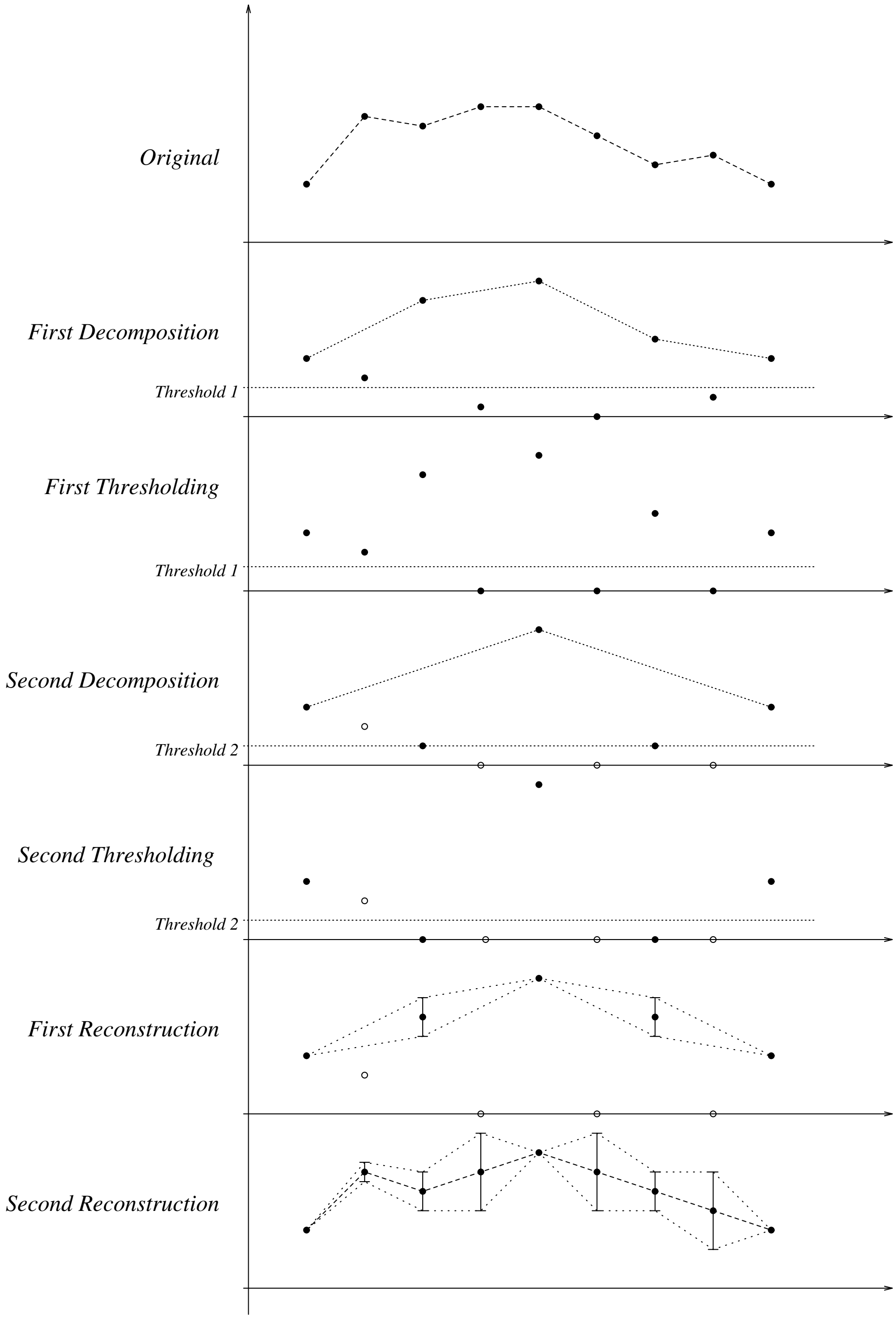


Figura G.5. Two iterations of the discussed procedure on a linear profile of an image. The active points are shown in black.

Figura G.6. Original Lena image: 512×512 8 bit/pixel. The lossless compression of this image with the LZW algorithm reaches about 6.67 bits/pixel.

Mapping a 2D Image to a 1D Data Stream

The mapping of the 2D image array to a stream of values is a critical phase, which determines the effectiveness of the subsequent lossless compression. With a LZW algorithm, the best compression can be reached when the data stream is formed by multiple occurrencies of the same chains of values; the longer the chains, the higher the compression. This feature is achieved by the JPEG standard with a *zig-zag* scanning of each sub-block in which the image is partitioned [106]. This specific scanning assures that in the final data stream similar values are grouped in nearby positions.

The traditional left-to-right and top-to-bottom scanning applied to the original image and to the image obtained by the discussed reorganization procedure provides two data streams whose compression is not sensibly different. This is due to the fact that the subimages which compose the reorganized image are interleaved. As an example, the first row of table G.1 shows the effect of the compression of the data streams generated by a left-to-right and top-to-bottom scanning on the original image (depicted in figure G.6) and on the images obtained after 1,2,3,4, and 5 lossless hierarchical decompositions. Figure G.8 shows the set of 7 images ($S^{(2)}$, $X^{(2)}$, $Y^{(2)}$, $Z^{(2)}$, $X^{(1)}$, $Y^{(1)}$, $Z^{(1)}$) produced by 2 iterations of the lossless decomposition. Nevertheless, the little improvement in the compression of the reorganized images (1,2,3,4, or 5 iterations) with respect to the compression of the original one (0 iterations) is due to the higher polarization of the image histogram around low values.

Different mappings $\mathcal{M}(\cdot) : \mathcal{F}^{k \times k} \rightarrow \mathcal{F}^{k^2 \times 1}$ have been considered and tested: their common underlying structure is aimed to the grouping of pixels with similar values in subsequent clusters. The one which has been proven to be the most effective is based on the left-to-right and top-to-bottom scanning of the subsampled image S first, followed by the contemporary scanning of the other 3 images $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$ one pixel each at a time. The contemporary scanning of images $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$ is justified by the fact that in correspondence of a significant brightness discontinuity all three images may present a non-zero value

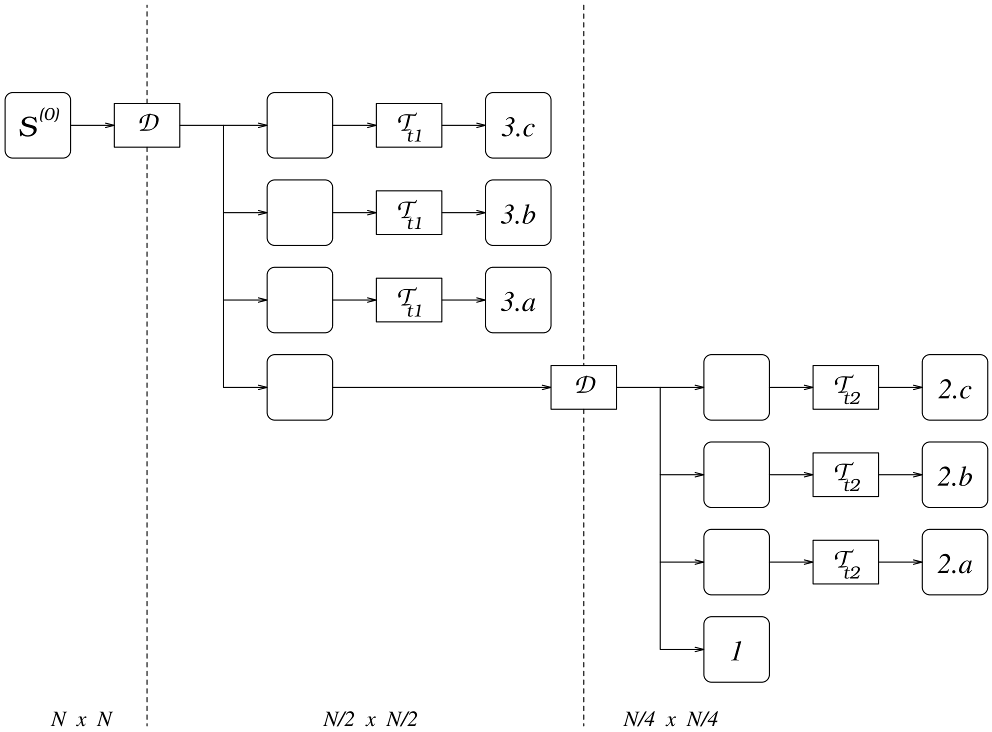


Figura G.7. Mapping between the 2D image array and a 1D data stream: first the image labeled with '1' is scanned in a left-to-right and top-to-bottom fashion; then images labeled with '2.a', '2.b', and '2.c' are contemporary scanned, one pixel at a time; the same is then iterated on images '3.a', '3.b', and '3.c'.

which has not been filtered by the threshold $\mathcal{T}_t(\cdot)$. In the same time, all the pixels with similar values (where 0 is a special case) are grouped together in subsequent clusters.

The specific mapping between the 2D image array and the 1D data stream can be extended to the hierarchical set of images generated by n iterations of the process. If n hierarchical decompositions have been made, the mapping is based on the scanning of $S^{(n)}$, followed by the contemporary scannings of $\left\{\tilde{X}^{(i)}, \tilde{Y}^{(i)}, \tilde{Z}^{(i)}\right\}$ with $i = n, ..., 1$. The second row of table G.1 shows the effect of the compression when the discussed 2D$\rightarrow$1D mapping is used. As an example, figure G.7 shows the order of scanning in the case of 2 hierarchical iterations.

Number of iterations	**Compression ratio** (standard scanning)	**Compression ratio** (proposed scanning)
0 (original)	6.67 bit/pixel	-
1	3.34 bit/pixel	3.10 bit/pixel
2	2.49 bit/pixel	2.23 bit/pixel
3	2.23 bit/pixel	2.04 bit/pixel
4	2.15 bit/pixel	2.00 bit/pixel
5	2.14 bit/pixel	2.00 bit/pixel

Tabella G.1. Effectiveness of the serial LZW compression of the output image obtained after 0 (original), 1, 2, 3, 4, and 5 iterations of the lossless decomposition. The values presented refer to an average computed on a set of standard test images (512×512, 8 bit/pixel).

A further advantage of the hierarchical approach allows its application to the transmission of images over different channels with different bandwidth. In fact, since the 1D data stream is formed by the concatenation of the sequence of the subsampled image $S^{(n)}$ and its local corrections $\tilde{X}^{(i)}$, $\tilde{Y}^{(i)}$, and $\tilde{Z}^{(i)}$, with $i = n, ...1$, this specific mapping allows:

1. to achieve a higher compression ratio, since similar values are grouped together, and
2. to start the reconstruction process before the transfer of the image has been completed.

The main advantage of point 2 lays in the possibility to recover a low resolution image first, and to increase its detail with the integration of the information which follow. Thus, as in the case of the MPEG-2 for image sequence compression, it allows to modify dynamically the resolution of the recovered image (or sequence of images) according to the channel bandwidth and/or computational power available.

G.2 Performance Analysis

The first version of this parallel preprocessing filter has been implemented on a Connection Machine CM-2 [62] for testing and tuning purposes. Currently it is under porting to the target architecture, PAPRICA, and the final implementation is foreseen in a couple of months.

This section presents the expected performances obtained in the processing of a 256×256 8/bit still image on PAPRICA architecture. The extremely simple set of operations that must be performed by each processor is listed in the following:

- determination of the average over the x, y, and xy directions in its 3×3 neighborhood;
- comparison with a fixed threshold;
- subsequent conditional reset of the values below the threshold.

All these operations must be iterated for each level of the hierarchical data set (pyramid). Contrary to general purpose parallel architectures (like the CM-2), pyramidal operations cause no computational overhead to PAPRICA system, as shown in section 3.5.

According to the theoretical analysis presented in [14], the PAPRICA processing of a $N \times N$ image with an algorithm based on the analysis of a 3×3 neighborhood (Moore Cellular Automaton [43, 46]) must be split in $\left(\left\lceil \frac{N-2}{Q-2} \right\rceil\right)^2$ subwindows, where Q represents the linear dimension of the PA.

Indicating with T_M the memory access time, and with T_C the PA clock cycle time, the time required to process a single image subwindow [14] is given by the sum of the loading time, processing time, and saving time. The loading and saving times can be considered equal and given by the number of pixels to be loaded ($Q \times Q$) times the memory access time.

$$T_{\text{loading}} = T_{\text{saving}} = Q^2 T_M = 16^2 \; 150 \text{ ns} = 38400 \text{ ns} . \tag{G.13}$$

The processing time is given by the number of assembly instructions executed by the PA (L, roughly 500) times the processors clock cycle time:

$$T_{\text{processing}} = L T_C \simeq 500 \; 250 \text{ ns} = 125000 \text{ ns} . \tag{G.14}$$

The total time required to process a single subwindow on the current version of the PAPRICA system is given by:

$$T_{\text{single window}} = T_{\text{loading}} + T_{\text{processing}} + T_{\text{saving}} \simeq 0.2 \text{ ms} , \tag{G.15}$$

which, multiplied by the number of subwindows in which a 256×256 image is partitioned becomes:

$$T_{\text{total}} = 0.2 \; 361 \simeq 73 \text{ ms} . \tag{G.16}$$

As an example, 4 hierarchical iterations on a 256×256 image require the processing of $361 + 81 + 25 + 9 = 476$ subwindows, and the total time becomes about 96 ms.

Finally, the time required for the whole acquisition, processing, and compression of a single still image is given by the following contributes:

- the acquisition time for a 256×256 image is 20 ms;
- the parallel preprocessing takes about 100 ms;
- the 2D$\rightarrow$1D mapping (basically the I/O time required to transfer the preprocessed image from PAPRICA system to its front-end) takes about 30 ms;

Operation	**Time required [ms]**
Acquisition time	$\simeq$ 20 ms
Data reorganization on PAPRICA	$\simeq$ 96 ms
Transfer time from PAPRICA system to its front-end processor	$\simeq$ 30 ms
Serial compression on PAPRICA front-end	in parallel with the acquisition and processing of the next frame

Tabella G.2. Expected performances of the complete system

- the serial compression of the 1D data stream is performed by PAPRICA front-end in parallel with the acquisition and preprocessing of the following frame by PAPRICA system, and thus the serial compression time is not considered.

The expected processing speed achievable with the discussed system is evaluated in about 6-7 frames/s; these performances are summarized in table G.2. Moreover, if the computing architecture could be able to load images in the image memory in parallel with the processing [40], the loading time could be neglected, and the processing speed could be furthermore increased.

The output quality of the discussed procedure has been compared to the standard JPEG compression technique (at the same compression ratio), and although the former looks better than the latter only in few cases, the extremely fast computation achievable on low cost and low power systems justifies the consideration and the study of this new technique.

Figure G.10 and figure G.11 show the quality of a 0.23 bit/pixel compression of figure G.6 with the discussed procedure (5 iterations with thresholds 100, 60, 15, 6, 0) and the standard JPEG technique respectively. Figure G.12 show an enlarged portion of the original image of figure G.6, while figures G.13 and G.14 show an enlarged portion of figures G.10 and G.11 respectively; the blocky effect, which represents the main artifact introduced by the JPEG compression (see figure G.14), is replaced by a more tolerable linear behavior (see figure G.13).

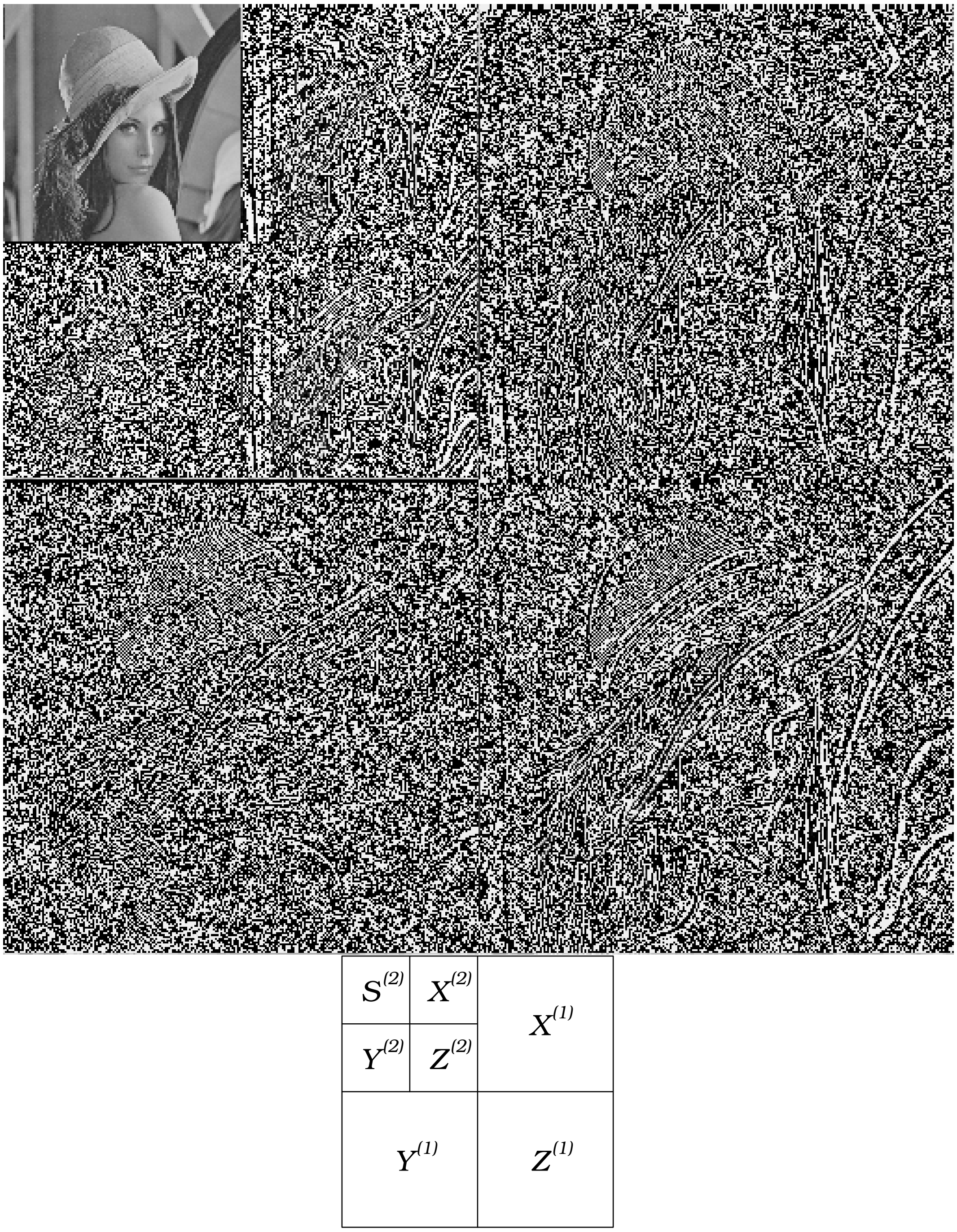


Figura G.8. Lena image after 2 hierarchical iterations of the lossless decomposition (512×512). The lossless compression of this image with the LZW algorithm reaches about 2.26 bits/pixel.

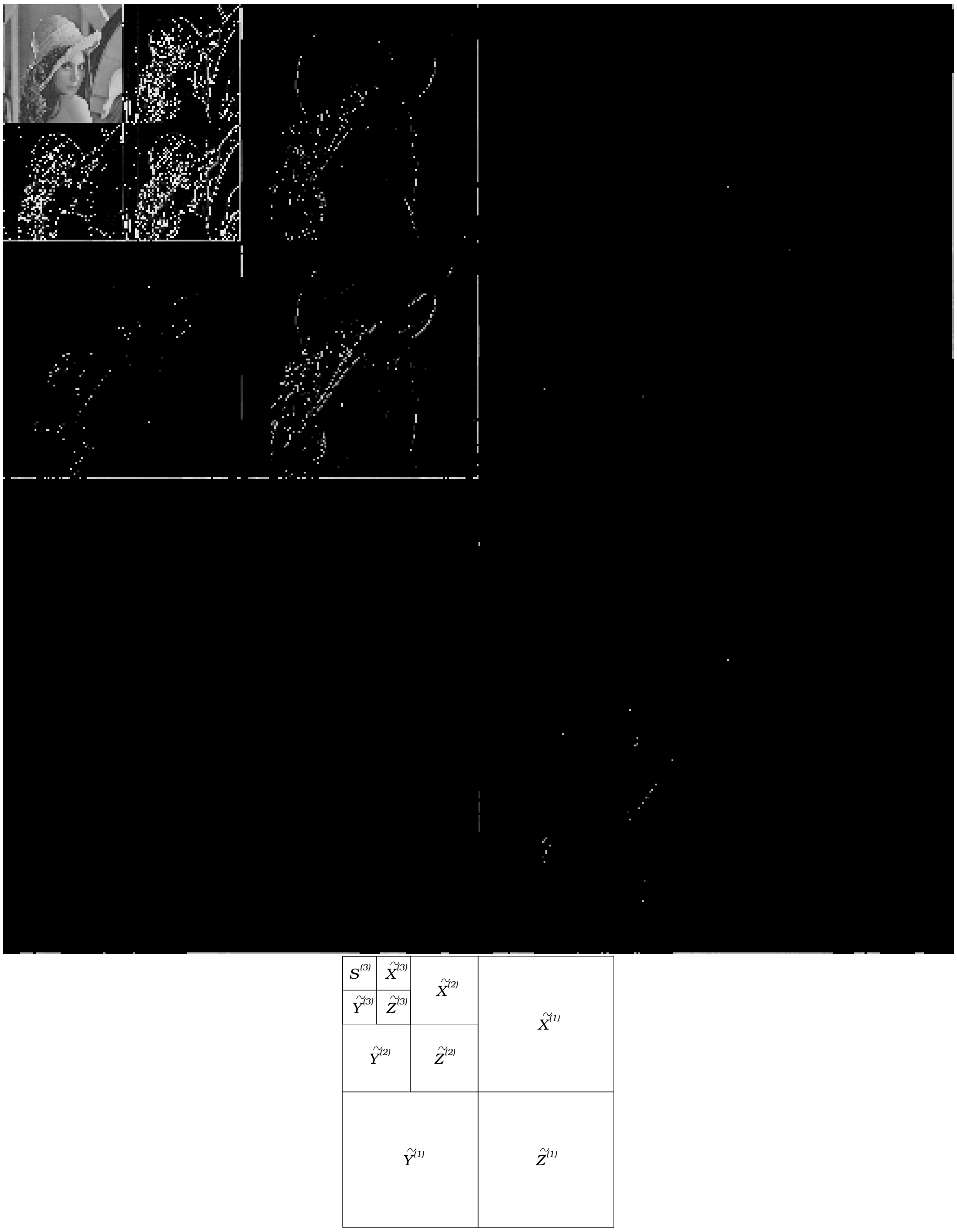


Figura G.9. Lena image after 3 hierarchical iterations of the lossless decomposition followed by the non-linear filter with thresholds 60, 40, and 10 (512×512). The lossless compression of this image with the LZW algorithm reaches about 0.43 bits/pixel.

Figura G.10. Lena image: 0.23 bit/pixel compression obtained with 5 hierarchical iterations of the discussed procedure (with thresholds 100, 60, 15, 6, and 0).

Figura G.11. Lena image: 0.23 bit/pixel compression obtained with the JPEG algorithm: edges are preserved better than in the previous image, but an annoying blocky effect appears

Figura G.12. Part of the original Lena image

Figura G.13. Part of Lena image: 0.23 bit/pixel obtained with the discussed procedure (5 iterations with thresholds 100, 60, 15, 6, and 0): the linear behavior is visible

Figura G.14. Part of Lena image: 0.23 bit/pixel obtained with the JPEG algorithm: the blocky effect is clearly visible

Bibliografia

[1] L. Abbot, R. M. Haralick, and X. Zhuang. Pipeline Architecture for Morphological Image Analysis. *International Journal of Machine Vision and Applications*, -(1):23–40, 1988.

[2] G. Adorni, A. Broggi, G. Conte, and V. D'Andrea. A self-tuning system for real-time Optical Flow detection. In *Proceedings IEEE System, Man, and Cybernetics Conference*, volume 3, pages 7–12, Le Toquet, France, October 17-20 1993. IEEE System, Man and Cybernetics.

[3] G. Adorni, A. Broggi, G. Conte, and V. D'Andrea. How does a Cellular Automaton perceive the Kanizsa Triangle? *Perception*, 22(Supplement):52, 1993.

[4] G. Adorni, A. Broggi, G. Conte, and V. D'Andrea. Massively Parallel Optical Flow computation for real-time vehicle navigation. In *Proceedings of the Second European CM users meeting*, Paris, October 11-14 1993. World Scientific. In press.

[5] G. Adorni, A. Broggi, G. Conte, V. D'Andrea, and C. Sansoè. High-level and low-level computer vision: towards an integrated approach. In E. Ardizzone, S. Gaglio, and F. Sorbello, editors, *Trends in Artificial Intelligence*, pages 322–331. Lecture Notes in Artificial Intelligence, Springer-Verlag, 1991.

[6] M. Albanesi, V. Cantoni, U. Cei, M. Ferretti, and M. Mosconi. Embedding Pyramids into Mesh Arrays. In H. Li and Q. Stout, editors, *Reconfigurable Massively Parallel Computers*, pages 123–140, New Jersey, 1991. Englewood Cliffs.

[7] C. Arcelli, L. Cordella, and S. Levialdi. Parallel Thinning of Binary Pictures. *Electronics Letters*, 11:148–149, July 1975.

[8] D. H. Ballard and C. M. Brown. *Computer Vision.* Prentice Hall, 1982.

[9] K. Batcher. Design of a Massively parallel processor. *IEEE Transactions on Computers*, C-29:836–840, 1980.

[10] B. Bhanu, P. Symosek, J. Ming, W. Burger, H. Nasr, and J. Kim. Qualitative target motion detection and tracking. In *Proceedings of DARPA Image Understanding Workshop*, San Mateo - California, May 1989. Morgan Kaufmann Pub. Inc.

[11] A. Blake and A. Zisserman. *Visual Reconstruction.* MIT Press, Cambridge MA, 1987.

[12] G. Borgefors. Distance Transformations in Digital Images. In *Computer Vision, Graphics and Image Processing*, volume 34, pages 344–371, 1986.

[13] W. L. Bouknight et al. The ILLIAC IV system. *IEEE Proceedings*, 60:369–388, 1972.

[14] A. Broggi. Performance Optimization on Low-Cost Cellular Array Processors. In *Proceedings MPCS - IEEE International Conference on Massively Parallel Computing Systems*, pages 334–338, Ischia, Italy, May 2-6 1994. IEEE Computer Society - EuroMicro.

[15] A. Broggi. Exploiting A-Priori Knowledge for Vision-Based Road Following Applications. Technical Report TR009-95, Dipartimento di Ingegneria dell'Informazione, Università di Parma, January 1995. Submitted to IJCAI'95.

[16] A. Broggi. Parallel and Local Feature Extraction: a Real-Time Approach to Road Boundary Detection. *IEEE Transactions on Image Processing*, February 1995. In press.

[17] A. Broggi. Word Parallelism vs Spatial Parallelism: a performance optimization technique on the PAPRICA system. In *Proceedings 3*$^{\text{rd}}$ *EuroMicro Workshop*

on Parallel and Distributed Processing, pages 236–243, Sanremo, Italy, January, 25-27 1995. IEEE Computer Society - EuroMicro.

[18] A. Broggi and S. Bertè. Riconoscimento guidato di bordi strada mediante Morfologia Matematica. Technical Report WP005-93, Dipartimento di Ingegneria dell'Informazione, Università di Parma, 1993.

[19] A. Broggi and S. Bertè. A Morphological Model-Driven Approach to Real-Time Road Boundary Detection for Vision-Based Automotive Systems. In *Proceedings Second IEEE Workshop on Applications of Computer Vision*, pages 73–90, Sarasota, FL, December, 5-7 1994. IEEE Computer Society.

[20] A. Broggi and S. Bertè. Real-Time Morphological Processing: an Application for Automated Navigation. In J. Serra and P. Soille, editors, *ISMM'94: Mathematical Morphology and its Applications to Image Processing*, pages 27–28, Fontainebleau, France, 1994. Ecole des Mines de Paris.

[21] A. Broggi, G. Conte, G. Burzio, L. Lavagno, F. Gregoretti, C. Sansoè, and L. M. Reyneri. PAPRICA-3: A Real-Time Morphological Image Processor. In *Proceedings ICIP - First IEEE International Conference on Image Processing*, pages 654–658, Austin, TX, November 13-16 1994. IEEE Computer Society.

[22] A. Broggi, G. Conte, F. Gregoretti, L. Reyneri, L. Rigazio, C. Sansoè, and C. Zamiri. PAPRICA. In *CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design*, pages 21–141. CNR - Progetto Finalizzato MADESS, Roma, Italy, 1990.

[23] A. Broggi, G. Conte, F. Gregoretti, C. Sansoè, and L. M. Reyneri. The PAPRICA Massively Parallel Processor. In *Proceedings MPCS - IEEE International Conference on Massively Parallel Computing Systems*, pages 16–30, Ischia, Italy, May 2-6 1994. IEEE Computer Society - EuroMicro. *Awarded by IEEE Computer Society as Best Paper presenting a European prototype of a massively parallel system.*

[24] A. Broggi and V. D'Andrea. Elaborazione d'Immagini Mediante Automi Cellulari: un Algoritmo per l'Estrazione di Caratteristiche. In *I Sensi dell'Automa*,

pages 245–257. LINT, Trieste, Italy, 1992.

[25] A. Broggi and V. D'Andrea. A Multiresolution Algorithm for Feature Tracking in Image Sequences. In S. Impedovo, editor, *Progress in Image Analysis and Processing III*, pages 218–221. World Scientific, 1993.

[26] A. Broggi, V. D'Andrea, and F. Gregoretti. A low-cost parallel VLSI architecture for low-level vision. In M. Takagi, editor, *MVA'92 - IAPR Workshop on Machine Vision and Applications*, pages 11–15, Tokyo, Japan, 1992. International Association for Pattern Recognition, IAPR.

[27] A. Broggi and A. Gandini. Parallel Image Clustering: A Real-Time Application for a Special-Purpose Architecture. In K. A. Hogda, B. Braathen, and K. Heia, editors, *Proceedings 8th SCIA - Scandinavian Conference on Image Analysis*, volume 1, pages 297–304, Tromsoe, Norway, May 25-28 1993. IAPR - International Association for Pattern Recognition, NOBIM - Norwegian Society for Image Processing and Pattern Recognition.

[28] A. Broggi and G. Gennari. Scomposizione di Matching Elements per PAPRICA. Technical Report WP004-93, Dipartimento di Ingegneria dell'Informazione, Università di Parma, September 1993.

[29] A. Broggi, S. Mora, and C. Sansoè. Enhancement of a 2D Array Processor for an Efficient Implementation of Visual Perception Tasks. In M. A. Bayoumi, L. S. Davis, and K. P. Valavanis, editors, *Proceedings CAMP'93 - Computer Architectures for Machine Perception*, pages 172–178, New Orleans, December 15-17 1993. IEEE Computer Society.

[30] A. Broggi, N. Patrignani, and A. Ambriola. Image Analysis on PAPRICA. In *CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design*, pages 21–141. CNR - Progetto Finalizzato MADESS, Roma, Italy, 1990.

[31] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, November 1986.

[32] V. Cantoni and M. Ferretti. *Pyramidal Architectures for Computer Vision*. Plenum Press, London, 1993.

[33] V. Cantoni, M. Ferretti, and M. Savini. Compact Pyramidal Architectures. In J. Tou and J. Balchen, editors, *Highly Redundant Sensing in Robotic Systems*, volume 58, pages 157–174. NATO ASI Series F, 1990.

[34] V. Cantoni and S. Levialdi. *Pyramidal Systems for Computer Vision*. Springer Verlag, Berlin, 1986.

[35] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-Power CMOS Digital Design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[36] E. F. Codd. *Cellular Automata*. Academic Press, New York, 1968.

[37] G. Conte, F. Gregoretti, L. Reyneri, and C. Sansoé. PAPRICA: a Parallel Architecture for VLSI CAD. In A.P.Ambler, P.Agrawal, and W.R.Moore, editors, *CAD Accelerators*, pages 177–189. North Holland, Amsterdam, 1991.

[38] B. Courtois. CAD and Testing of ICs and systems: Where are we going? Technical report, TIMA & CMP, January 1993.

[39] S. J. Crabtree, L.-P. Yuan, and R. Ehrlich. A Fast and Accurate Erosion-Dilation Method Suitable for Microcomputers. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 53(3):283–290, May 1991.

[40] R. Cucchiara. *Parallelismo Spaziale e Temporale nelle Architetture per l'Elaborazione di Immagini*. PhD thesis, DEIS, Università degli Studi di Bologna, Bologna, Italy, 1993.

[41] R. Cucchiara, T. S. Cinotti, G. Neri, and G. Rustichelli. Reconfiguring the boundaries of a mesh-connected array of processors with run-time programmable logic. *Microprocessors and Microsystems*, 17(2):67–73, 1993.

[42] R. Cucchiara, M. Piccardi, L. D. Stefano, M. Monacelli, G. Rustichelli, and T. S. Cinotti. The Protorype of a Heterogeneous Machine for Robot Vision: Architecture, Programming and Performance. In *PSF94 - Parallel System Fair 94 - IEEE International Symposium of Parallel and Distributed Systems*, Cancun, Mexico, April 1994.

[43] K. Culik II, L. P. Hurd, and S. Yu. Computation theorethic aspects of Cellular Automata. *Physica D*, 45:431–440, 1990.

[44] M. Duff. Parallel Processors for Digital Image Processing. In P. Stucki, editor, *Advances in Digital Image Processing*, pages 265–279. Plenum Press, London, 1979.

[45] M. Duff, D.M.Watson, T. Fountain, and G. Shaw. A Cellular Logic array for Image Processing. *Pattern Recognition*, 15:229–247, 1973.

[46] D. Farmer, T. Toffoli, and S. Wolfram, editors. *Cellular Automata*, Amsterdam, March 1983. North-Holland.

[47] F.Gerritsen and P.Verbeck. Implementation of cellular logic operators using 3×3 convolution and table lookup hardware. *Computer Vision, Graphics and Image Processing*, 27, 1984.

[48] G. H. Forman and J. Zahorjan. The Challenge of Mobile Computing. *Computer*, 27(4):38–47, April 1994.

[49] T. Fountain. *Processor Arrays: Architectures and applications.* Academic-Press, London, 1987.

[50] T. Fountain and V. Goetcherian. CLIP4 Parallel Processing System. *IEE Proceedings*, 127E:219–224, 1980.

[51] T. Fountain and K. Matthews. The CLIP 7A Image Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):310–319, May 1988.

[52] M. Garzon. Cellular Automata and discrete neural networks. *Physica D*, 45:431–440, 1990.

[53] D. Geiger and F. Girosi. Parallel and Deterministic Algorithms for Markov Random Fields: Surface Reconstruction and Integration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:401–412, 1991.

[54] V. Graefe and K.-D. Kuhnert. Vision-based Autonomous Road Vehicles. In I. Masaki, editor, *Vision-based Vehicle Guidance*, pages 1–29. Springer Verlag, 1991.

[55] F. Gregoretti, L. M. Reyneri, C. Sansoè, A. Broggi, and G. Conte. PAPRICA - A Dedicated Processor for Morphological Image Analysis. In S. Impedovo, editor, *Progress in Image Analysis and Processing III*, pages 721–724. World Scientific, 1993.

[56] F. Gregoretti, L. M. Reyneri, C. Sansoè, A. Broggi, and G. Conte. PAPRICA: an integrated approach to hierarchical morphology. Technical Report TR001-93, Dipartimento di Ingegneria dell'Informazione, Università di Parma, February 1993.

[57] F. Gregoretti, L. M. Reyneri, C. Sansoè, A. Broggi, and G. Conte. The PAPRICA SIMD array: critical reviews and perspectives. In L. Dadda and B. Wah, editors, *Proceedings ASAP'93 - IEEE Computer Society International Conference on Application Specific Array Processors*, pages 309–320, Venezia, Italy, October 25-27 1993. IEEE Computer Society - EuroMicro.

[58] F. Gregoretti, L. M. Reyneri, C. Sansoè, and L. Rigazio. A chip set implementation of a parallel cellular architecture. *Microprocessing and Microprogramming*, 35:417–425, 1992.

[59] F. C. A. Groen and N. J. Foster. A fast algorithm for cellular logic operations on sequential machines. *Pattern Recognition Lett.*, 2:333–338, 1984.

[60] L. G. C. Hamey, J. A. Web, and I. Wu. Low-level vision on Warp and the Apply Programming Model. In J. S. Kowalik, editor, *Parallel Computations and Computers for Artificial Intelligence*, pages 185–200. Kluwer Academic Publishers, 1988.

[61] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image Analysis Using Mathematical Morphology. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 9(4):532–550, 1987.

[62] W. D. Hillis. *The Connection Machine.* MIT Press, Cambridge, Ma., 1985.

[63] I.N.Robinson and W.R.Moore. A parallel processor array architecture and its implementation in silicon. In *Proceedings of IEEE Custom Integrated Circuits Conference*, pages 41–45, New York, 1982. Rochester.

[64] T. M. Jochem and S. Baluja. A Massively Parallel Road Follower. In M. A. Bayoumi, L. S. Davis, and K. P. Valavanis, editors, *Proceedings CAMP'93 - Computer Architectures for Machine Perception*, pages 2–12, New Orleans, December 15-17 1993. IEEE Computer Society.

[65] T. M. Jochem and S. Baluja. Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following. In H. Kitano, editor, *Massively Parallel Artificial Intelligence.* AIII Publishers in cooperation with MIT Press, 1993.

[66] T. M. Jochem, D. A. Pomerleau, and C. E. Thorpe. MANIAC: A Next Generation Neurally Based Autonomous Road Follower. In *Proceedings of the International Conference on Intelligent Autonomous Systems: IAS -3*, Pittsburgh, Pennsylvania, USA, February 1993.

[67] M. Kimmel, R. Jaffe, J. Manderville, and M. Lavin. MITE: Morphic Image Transform Engine, an architecture for reconfigurable pipelines of neighborhood processes. In *IEEE Computer Society Workshop for Pattern Analysis and Image Database Management*, pages 493–500, Miami Beach, Florida, November 1985.

[68] J. Klein and J. Serra. The texture analyser. *Journal of Microscopy*, 95(2):349–356, April 1973.

[69] N. Margolus and T. Toffoli. Cellular Automata Machines. In G. D. Doolen et al., editors, *Lattice Gas Methods for Partial Differential Equations*, pages 219–249, Redwood City, California, 1990. Addison Wesley.

[70] MasPar Computer Corporation, Sunnyvale, California. *MP-1 Family Data-Parallel Computers*, 1990.

[71] G. Matheron. *Random Sets and Integral Geometry.* John Wiley, New York, 1975.

[72] D. McCubbery and R.Lougheed. Morfological Image Analysis Using a Raster Pipeline Processor. In *Workshop Computer Architecture for Pattern Analysis and Image Database Management.* IEEE Computer Society, November 1985.

[73] M. Mordonini. Compressione di immagini mediante tecniche massivamete parallele. Master's thesis, Università degli Studi di Parma - Facoltà di Ingegneria, 1994.

[74] NCR Corporation, Dayton, Ohio. *Geometric Arithmetic Parallel Processor*, 1984.

[75] H. Neumann and H. Stiehl. Toward a computational architecture for monocular preattentive segmentation. In R. G.Hartmann and G.Hauske, editors, *Parallel Processing in Neural Systems and Computers.* Elsevier (North Holland), 1990.

[76] W. M. Newman and R. F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, Tokyo, 1981.

[77] I. Pitas and A. N. Venetsanopoulos. Morphological Shape Decomposition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 14(1):38–45, January 1990.

[78] D. A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Kluwer Academic Publishers, Boston, 1993.

[79] W. K. Pratt. *Digital Image Processing.* John Wiley & Sons, 1978.

[80] K. Preston. The Abingdon Cross Benchmark Survey. *Computer*, pages 9–18, July 1989.

[81] S. Reddaway. DAP - A Distributed Array Processor. In *1st Annual Symposium on Computer Architectures*, pages 61–65, Florida, 1973.

[82] C. H. Richardson and W. Schafer. A Lower Bound for Structuring Element Decomposition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(4):365–369, April 1991.

[83] R.M.Lougheed and D.L.McCubbrey. The Cytocomputer: A Practical Pipelined Image Processor. In *7th Annual Symposium on Computer Architectures*, pages 271–277, 1980.

[84] A. Rosenfeld. A characterization of parallel thinning algorithms. *Information Control*, 29:286–291, 1975.

[85] A. Rosenfeld. *Multiresolution Image Processing and Analysis.* Springer Verlag, Berlin, 1984.

[86] U. Schmidt and K. Caesar. Datawave: A single-chip multiprocessor for video applications. *IEEE Micro*, pages 22–94, June 1991.

[87] L. A. Schmitt and S. S. Wilson. The AIS-5000 Parallel Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):320–330, May 1988.

[88] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, London, 1982.

[89] F. Y.-C. Shih and O. R. Mitchell. Decomposition of gray-scale morphological structuring elements. *Pattern Recognition*, 24(3):195–203, 1991.

[90] M. Shoji. *CMOS Digital Circuit Technology.* Prentice Hall, 1988.

[91] S. R. Sternberg. An overview of image algebra and related architectures. In *Integrated technology for parallel image processing*, pages 79–100. Academic Press, London, 1985.

[92] S. L. Tanimoto and K. Kilger. *Structured Computer Vision: Machine Perception trough Hierarchical Compuation Structures.* Academic Press, NY, 1980.

[93] S. L. Tanimoto, T. J. Ligocki, and R. Ling. A Prototype Pyramid Machine for Hierarchical Cellular Logic. In *Parallel Computer Vision*. Academic Press, London, 1987.

[94] Thinking Machines Corporation, Cambridge, Ma. *Connection Machine CM-200 Series - Technical Summary*, 1991.

[95] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. The New Generation System for the CMU Navlab. In I. Masaki, editor, *Vision-based Vehicle Guidance*, pages 30–82. Springer Verlag, 1991.

[96] M. Tistarelli and G. Sandini. Dynamic aspects in active vision. *CVGIP: Image Understanding*, 56(1):108–129, July 1992.

[97] T. Toffoli. Cellular Automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D*, 10:117–127, 1984.

[98] T. Toffoli and N. Margolous. *Cellular Automata Machines*. MIT Press, Cambridge, MA, U.S.A., 1987.

[99] R. Tsai. An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. In *Proceedings IEEE International Conference on Computer Vision and Pattern Recognition*, pages 364–374, Miami Beach, FL, 1986.

[100] T.Sudo, T.Nakashima, M.Aoki, and T.Kondo. An LSI adaptive array processor. In *Proceedings IEEE International Solid-State Circuits Conference*, pages 122,123,307, 1982.

[101] S. Unger. A computer oriented toward spatial problems. *Proceedings IRE*, 46:1744–1750, 1958.

[102] R. van den Boomgaard. *Mathematical Morphology: Extensions Towards Computer Vision*. PhD thesis, Universiteit Van Amsterdam, 1992.

[103] R. van den Boomgaard and R. van Balen. Fast Algorithms for Morphological Image Operations using Bitmapped Binary Images. In B. Zavidovique and

P.-L. Wendel, editors, *Computer Architectures for Machine Perception*, pages 453–460, Paris, December 1991.

[104] R. van den Boomgaard and R. van Balen. Methods for Fast Morphological Image Transforms Using Bitmapped Binary Images. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 54(3):252–258, May 1992.

[105] B. J. H. Verwer and L. J. van Vliet. A contour processing method for fast binary neighborhood operations. *Pattern Recognition Lett.*, 7:27–36, 1988.

[106] G. M. Wallace. The JPEG Still Picture Compression Standard. *Communication of ACM*, April 1991.

[107] T. A. Welch. A Technique for High Performance Data Compression. *Computer*, 17(6):8–19, June 1984.

[108] S. S. Wilson. Theory of Matrix Morphology. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 14(6):636–652, June 1992.

[109] J. M. Wolfe and K. R. Cave. Deploying visual attention: the guided model. In *AI and the eye*, pages 79–103. A.Blake and T.Troscianko, 1990.

[110] J. Xu. Decomposition of Convex Polygonal Morphological Structuring Elements into Neighborhood Subsets. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(2):153–162, February 1991.

[111] X. Zhuang. Decomposition of Morphological Structuring Elements. Technical report, Department of Electrical and Computer Engineering, University of Missouri-Columbia, 1992.

[112] X. Zhuang and R. M. Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, and Image Processing*, 35:370–382, September 1986.